MS-DOS®
データブック

MSDOS.SYSの仕組みと
解析フローチャート

幸田敏記 著

ラジオ技術社

ラジオ技術選書181

MS-DOS® データブック

MSDOS.SYSの仕組みと解析フローチャート

幸田敏記 著

ラジオ技術社

本書で解説した*MS-DOS*® 2.11 バージョンは，下記米国マイクロソフト社によって製作されたものです．

Microsoft Corporation
16011 NE 36th Way
Box 97017
Redmond, WA 98073-9717

このプログラムの著作権は，すべて米国マイクロソフト社に帰属するものであり，この複製物を許諾（ライセンス契約）なく使用しまたは複製することは，法律に違反する行為となります．

本書を刊行するにあたり，米国マイクロソフト社の承認を得て***MS-DOS***®の解析を行いフローチャートの記載を行いました．このフローチャートは，***MS-DOS***®のプログラムの各ルーチン等と１対１の対応関係を有するものであり，***MS-DOS***®のプログラムから複製されたものであります．したがって，***MS-DOS***®と同一または類似の機能を持ち，本書記載のフローチャートと同一または類似の構造を有するプログラムの開発は，米国マイクロソフト社の***MS-DOS***®および本書のフローチャートの著作権を侵害することになります．

著　者　　幸 田 敏 記
株式会社　ラジオ技術社

著者のことば

MS-DOSのファンクションコールの中には，一般ユーザーに解放されてはいないがMS-DOSの外部コマンドの中では使用されているシステム用ファンクションコールがあります．また，FCB, PSP, ディレクトリエントリなどのシステムとのインタフェースを司るワークエリアの中にも，システムリザーブ領域としてシステム内で使用している部分があります．本書では，おもにこれらの，いままでまとまっては公表されなかったシステム領域に照準を合わせ解説しました．MS-DOSは，IO . SYS, MSDOS . SYS, COMMAND . COMからなっていますが，本書では， MSDOS . SYSを解析しました．COMMAND . COMも，その他の外部コマンドといっしょに続編で解析できるようになれば，と考えています．

第1章では，MS-DOSの概要から，リセットボタンを押して COMMAND . COM が立ち上がるまでの動作の概要を解説しています．ここでは，CONFIG . SYSがいかにして，デバイスドライバをシステムに登録するかがわかります．

第2章では PC-9801 につきには IO . SYSの仕組みと MSDOS.SYS との関連を説明しましたが，デバイスドライバの設計のさいの参考資料となるでしょう．

第3章では，INT 21H ファンクションコールの解析をしました．いままで公開されているファンクションコールについてはもちろん，未公開のファンクションコールについてもその使い方から処理の内容まで，フローチャートをまじえて解析してあります．MSDOS . SYSの解析は，通り一遍の解説書ではわかりにくかったファンクションコールの使い方も，解析を通してパラメータの意味するところやファンクションの処理などから浮き彫りにされることでしょう．具体的にいえば，ファンクション48Hの“MCBの割り当て”は，マニュアルなどの説明ではわかりにくいところですが，処理のフローチャートを追っていけば，使い方を明確に知ることができます．

第4章では，ちょっとした応用例を示しています．たとえば，「N88日本語BASIC(86)MS-DOS版」でMS-DOSのファンクションコールを使って20行モードでライン入力を実行する方法や，クラスタ値→セクタ値変換などです．とくに前者の例は，「N88日本語BASIC (86) MS-DOS版」でVJEやATOKなどの日本語入力フロントエンドプロセッサに対応するうえでのヒントとなるでしょう．本書のデータを有効に使用する上での参考にしてください．

第5章では，MS-DOSシステム解析に役立つ便利なプログラムを掲載しました．実際のディスクバッファや内部FCB, メモリコントロールブロック(MCB)，デバイスパラメータブロック(DPB)などのシステムエリアの内容を表示するものです．実際に走らせてみて，自分の手で確かめてください．また，これらのプログラムを参考にして，独自のユーティリティを作るのも勉強になるでしょう．

ただし注意してもらいたいのは，汎用性を考えるならあまり直接IO . SYSにアクセスしないことです．またMS-DOSの上位バージョンになったらなくなったかとか，機能が変更になったとかいうようなファンクションコールを使用しないことが大切でしょう．それらの見切りを付けるためにも，本書はデータブックとして，非常に役立つことでしょう．

MS-DOSの解析を思い立ったのは，マニュアルがわかりにくく，解説書を読んでもマニュアルの域を出ていないものばかりだったことがきっかけです．それでは，実際に動作しているMS-DOS自体に尋ねてみようと思ったのです．また，外部コマンドなどで使用している，マニュアルにないファンクションコールについて調べてみたいということもありました．解析するにつれてPSPや

FCBのシステムリザーブ領域となっているところにもある程度決まった働きをするワークエリアがあることもわかりました．オブジェクトプログラムからその働きを類推するのは，化石を掘り起こしているような，あるいはアドベンチャーゲームをしているようなおもしろさがあります．マシン語のオブジェクトプログラムを解析するには，ディスアセンブルリストを解読するのはもちろん，実際のMS-DOSの動作や仕様から内容を類推しなければなりません．あるときは，ダミーのデバイスドライバやINTルーチンのトラップをしかけ，MS-DOSの反応を見ることもあります．

パーソナルコンピュータはその名前のごとく，ひとりに1台というべきものなのですから，だれでもプログラムや拡張ボードを作れるようにしておけば，それらの有益なプログラム等の数は，それを使っている人の数だけ現われる可能性があります．そうはいかないまでも，そのパーソナルコンピュータのメーカーだけから供給される数より，サードパーティからのものも合わせれば，格段に大きくなります．また，すぐれたものが供給される可能性もふえるでしょう．そうすれば，それだけ，そのパーソナルコンピュータの価値は上がったことになるのです．あまり全部を公開するのも問題でしょうが，それより秘密主義に凝り固まってしまうことのほうが，パーソナルコンピュータに関しては，もっと危険であるといわねばならないでしょう．これから，パーソナルコンピュータの機能が巨大化していくなかで，そのようなことにならないように願っています．

なお，本書に掲載されたフローチャートその他の解析情報の掲載については，Microsoft社の了承を得ていますが，その解析は筆者独自のものですので，本書の内容についてのMicrosoft社への直接の問い合わせは御遠慮ください．

コツコツと解析してきたMS-DOS (Version 2.11) の成果を，ここにMicrosoft社の同意を得て発表することができて，非常にうれしく思います．

最後に，いろいろとお世話になった，アクセスグループの藤田英時さん，ラジオ技術社の鈴木勇治さん，また，執筆の間になにかと苦労をかけた，妻の裕子に感謝いたします．

昭和62年6月7日

幸田敏記（こうだ　としき）

MS-DOS® データブック

MSDOS.SYSの仕組みと解析フローチャート

第1章　システムの立ち上げの仕組み

第2章　IO. SYS の仕組み

第3章　MSDOS.SYS の仕組みと解析フローチャート

第4章 応用プログラム

第5章　解析に役立つユーティリティ

コラム

第1章

システムの立ち上げの仕組み

1・1 MS-DOSの概要

一口にMS-DOSといっても，大きく分けて3つの部分からなっています．

IO.SYS
MSDOS.SYS
COMMAND.COM

の3つです．これらのプログラムは，システムディスクに上記の名前で記録されています．IO.SYSとMSDOS.SYSはhidden（ヒドゥン）属性といって，DIRコマンドでは表示されません．

これらのシステムプログラムはシステムの立ち上げ時に（リセットボタンを押すと）順番にメモリにロードされますが，システムロード完了時のメモリ配置（メモリマップ）は**図1•1**のようになります．

それぞれのプログラムの働きは大まかにいうとつぎのようになります．

① **IO.SYS**：

ハードウェアを操作します．直接I/O (Input/Output) ポートをアクセスしたり，ROM-BIOSを使ったりして，ハードウェアとMSDOS.SYSの間のインタフェースをとりもちます．

具体的には，フロッピーディスクやハードディスクの入出力等があります．

コンソールにはカナ漢字変換の，いわゆる日本語入力の機能も含まれます．

IO.SYSはOEMメーカー（PC-9800シリーズでいうならNEC）が作成します．

② **MSDOS.SYS**：

MS-DOSの本体とでもいうべき部分です．おもにINT 21Hのファンクションコールをサポートします．すなわち，COMMAND.COMや外部コマンドからファンクションコールを受け取り，IO.SYSを呼び出すなどしてファンクションコールの機能をはたします．

IO.SYSはコマンドパケットという領域を用意して呼び出さなければならず，また，論理装置コードや物理セクタなどというハードに直接関係したものを扱わねばならないなど，少々手がかかりますが，MSDOS.SYSはファイル名でファイルのオープン，クローズの操作ができるなど，IO.SYSより一段高いインタフェースを持っています．

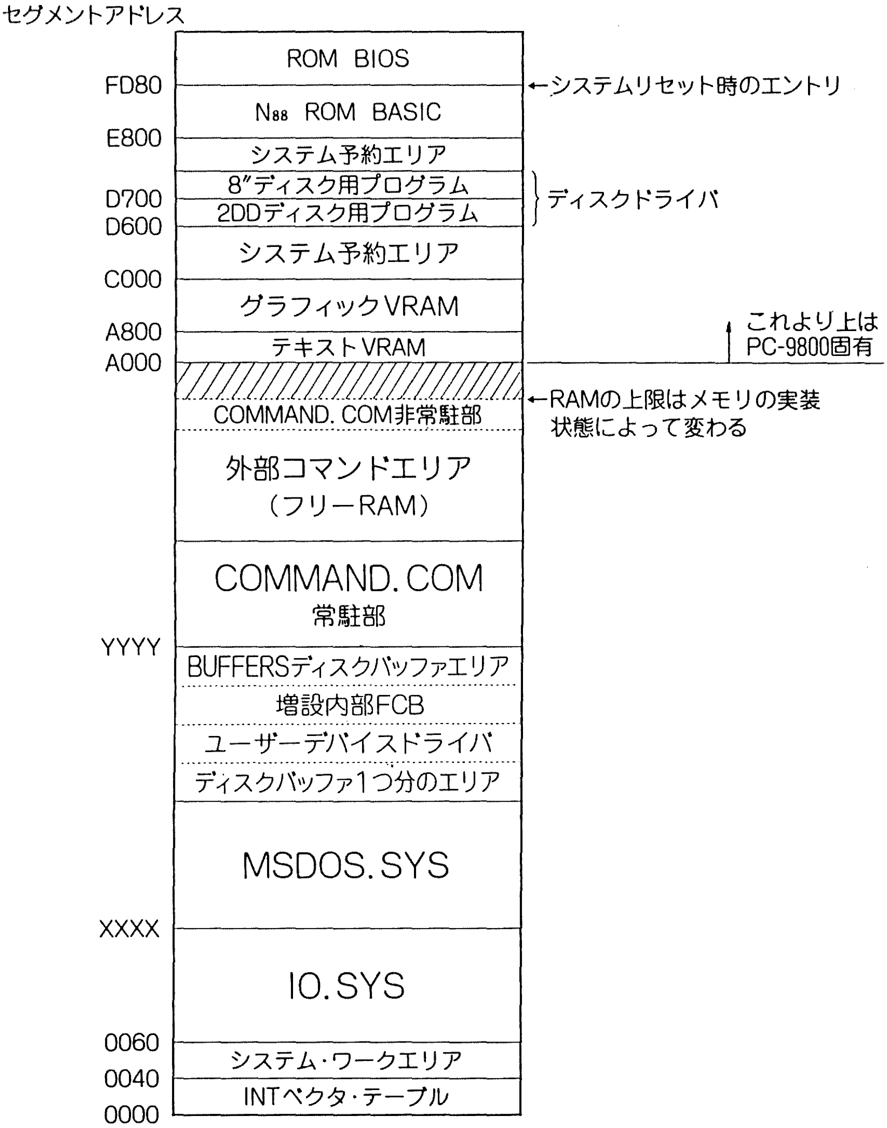
MSDOS.SYSはバージョン番号が同じなら各機種によって共通性が高いので，この本の解析で述べることは他の機種のMS-DOSでも非常に参考になるでしょう．

テンプレート機能もMSDOS.SYS内で処理しています．

③ **COMMAND.COM**：

コンソールから入力されるコマンドを実行します．いわゆる内部コマンドはCOMMAND.COMの中のサブルーチンになっていて，内部コマンドにないコマンドは外部コマンドとしてディスクからメモリにロードし実行します．

COMMAND.COMは常駐部と非常駐部とに分かれてメモリに配置され（**図1・1**），常駐部には致命的エラー処理ハンドラや外部コマンド実行ルーチン，非常駐部再入力ルーチンがあります．



IO.SYSのセグメントアドレスは60Hに固定
XXXX：MSDOS.SYSのセグメントアドレスは
　　　セグメント0000オフセット86, 87Hの内容
　　　（すなわちINT21Hファンクションコールの
　　　ベクタのセグメント部）
YYYY：COMMAND.COMのセグメントアドレスは
　　　セグメント0000オフセットBA, BBHの内容

図 1・1 **MS-DOS** システムメモリマップ

内部コマンドは COMMAND . COM の非常駐部にまとめられています。

バッチ処理やパイプ処理，環境などもここでサポートします。

同じ Ver. 2. xx でもいくつかの改良版[注]が見られますが，全体的な処理の流れや内部コマンド自体はほとんど共通しています。

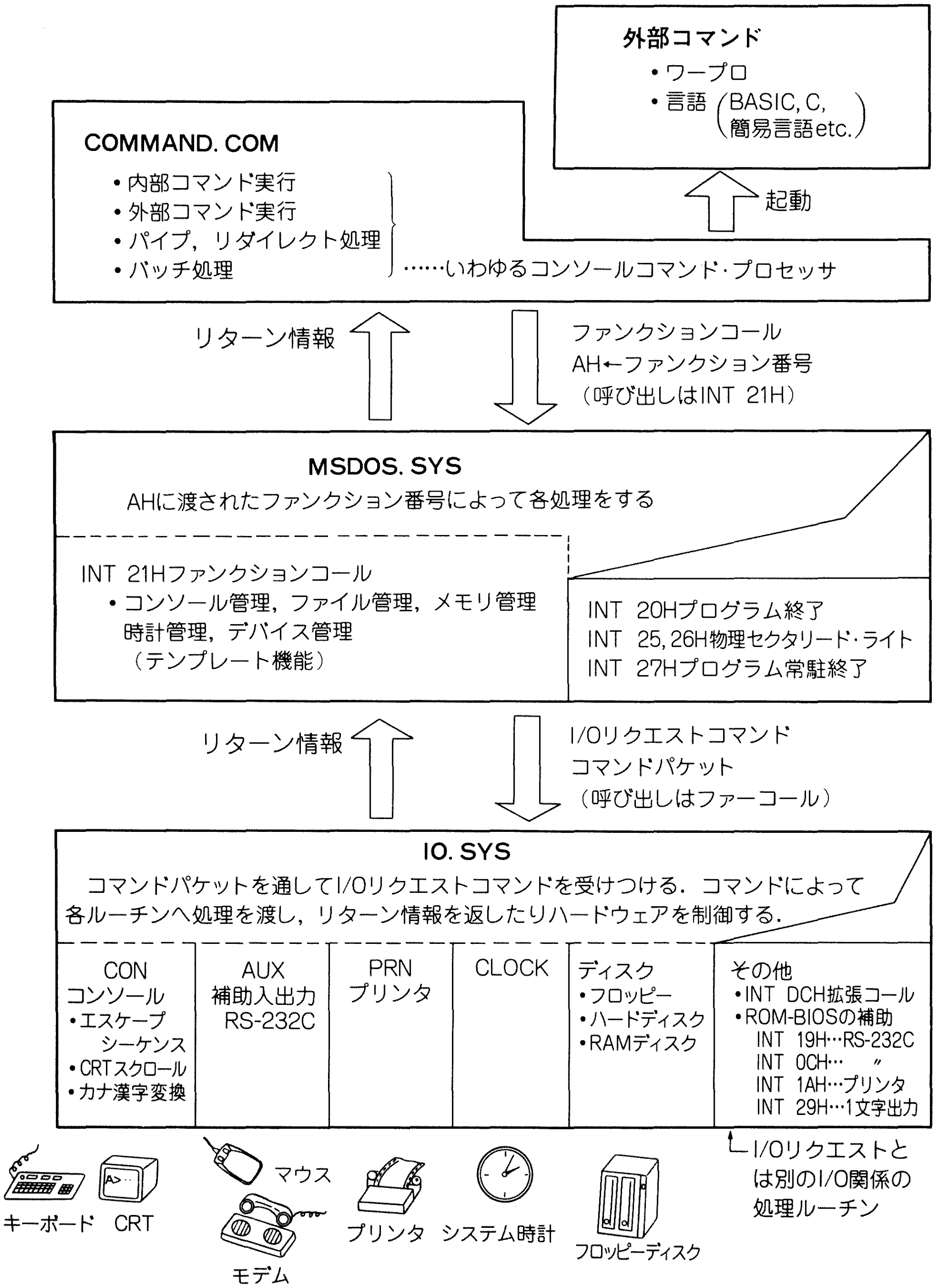


図 1・2 システムプログラム間の関係

IO . SYS, MSDOS . SYS, COMMAND . COM の間のデータやコマンドの流れは**図 1・2** に示すとおりです．

注） 外部コマンドの実行にパス名がつけられるものがあったり，FOR コマンドの内部バッファがオーバフローして処理が正常に実行されないものがある．

1・2 システムディスク

システムディスクは**図1・3**のように最初の1セクタがIPL, つぎにFAT, ディレクトリ, IO.SYS, MSDOS.SYSのプログラム, そのあとをフリーエリアとして使用するようになっています. IPLを格納する部分（FATより前のセクタ）をMS-DOSでは予備のセクタと呼んでいます.

システムディスクはSYS. COMコマンドかFORMAT. COMコマンドに/Sパラメータをつけて起動することにより作成できます. システムディスクのディレクトリは**図1・4**のようにディレクトリ先頭から「ボリュームラベル[注)], IO.SYS, MSDOS.SYS」または「IO.SYS, MSDOS.SYS」の順番でなければならないので, すでに他のファイルがディレクトリに登録されている状態ではシステムディスクを作成できないようになっています. IO.SYS, MSDOS.SYSが上記のように作成できることがわかると, ディレクトリのすぐうしろからIO.SYS, MSDOS.SYSプログラムが書き込

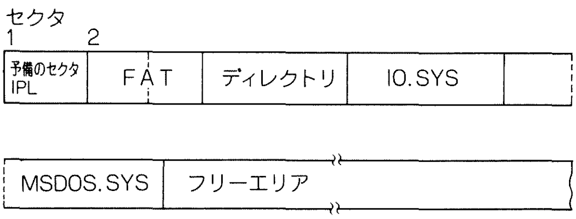


図 1・3　システムディスクのセクタ割り付け

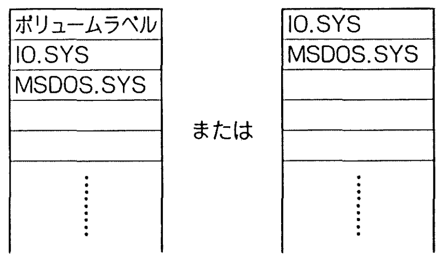


図 1・4　システムディスクのディレクトリ

FORMAT. COMのパラメータ

FORMAT. COMにはシステムディスクを作る/S　パラメータとボリュームラベルをつける/Vパラメータがありますが, この他にも/9, /C, /H, /Oというパラメータがあります.

/9……5”　2DDディスクにおいて1トラックを9セクタでフォーマットする. /9をつけなければ8セクタ/トラックになる.

8セクタ/トラック……全ディスク容量640kバイト
9セクタ/トラック……全ディスク容量720kバイト
(1kバイト＝1024バイト)
というように, 80kバイトのディスク容量の増加となる.

/C……FATとディレクトリの初期化のみをし物理フォーマットはしない. すでに物理フォーマットをしているディスクに対してのフォーマットに使い, 物理フォーマットをしないので速い.

/H……ハードディスクのフォーマットをする.

/O……ルートディレクトリにファンクションコールの16H（ファイルクリエイト）を使って, ディレクトリがいっぱいになるまでファイルを作ったのち削除する. ルートディレクトリに傷がないかどうかのチェックができる.

注)　ボリュームラベルはディレクトリの先頭のほう（2セクタ以内程度）にあれば, **図1・4**のようにディレクトリトップになくてもMS-DOSに認識されるようです（DIRをとったときにボリュームラベルが表示される).

まれます.

FORMAT. COM で/S を指定しないとデータディスクが作成されます. データディスクは**図1・3**で IO.SYS, MSDOS.SYS がなく，ディレクトリのすぐうしろからのデータフリーエリアがシステムディスクより大きくなります.

しかし，システムディスクをドライブ1に入れて電源を入れリセットボタンを押せば，MS-DOS システムを起動することができますが，データディスクでは 'This is the data disk' と画面に表示されてホールト（CPU が実行を停止）してしまいます.

1・3 システムの立ち上げ

MS-DOS システムがディスクからメモリに読み込まれ，COMMAND.COM が起動してプロンプト入力待ちになるまでを，駆け足で見ていきましょう.

1・3・1 ブートストラップローダ

電源を入れるか，電源を入れた状態でリセットボタンを押すと 8086CPU はリセットされ，FFFF：0000H から実行し始めます. このとき CS レジスタ以外のレジスタ類はすべて 0000H です. PC-9801 では FFFF：0000H は ROM エリアであり（**図1・1**）

EA 00 00 80 FD (JMP FAR FD80H：0000H)

という命令が記録されています. FD80：0000H も ROM エリアであり（**図1・1**），ROM BIOS の初期化ルーチンのエントリとなっています. ROM BIOS が使えるようになった状態で，ブートストラップローダが実行されます.

ブートストラップローダはディスクのトラック0，サーフェス0，セクタ1に書かれている IPL と呼ばれるプログラムを RAM 上の 1FC0：0000H に読み込み，その先頭アドレスへファージャンプします.

1・3・2 IPL（Initial Program Loader）

MS-DOS のディスクの0トラック，1セクタ目には IPL（Initial Program Loader）と呼ばれるものが記録されています. IPL はシステム起動時にディスクから IO.SYS と MSDOS.SYS プログラムを読み込むための短いプログラムです. IPL の大きさはふつう1セクタです.

IPL は大きく分けてデータディスク用とシステムディスク用があります.

① データディスク用 IPL

データディスク用 IPL は 'This is the data disk' と CRT 左上に表示してブザーを鳴らしホールト（CPU が実行を停止）してしまいます.

② システムディスク用 IPL

ディスクの型式（8インチ型か5インチ型かハードディスクか）によっていくつか用意されていま

す．PC-9801 では 8 インチ・フロッピー型に 1 つ，5 インチ・フロッピー 2DD 型に 2 つ（8 セクタ/トラック・フォーマット用と 9 セクタ/トラック・フォーマット用），ハードディスク型に 1 つ用意されています．

システムディスク用 IPL は，まずディレクトリ・トップセクタをメモリ 0060：0000H～に読み込んで IO．SYS と MSDOS．SYS がディレクトリに記録されているかを確かめます．このときディレクトリの先頭は，**図 1･4** のようにボリュームラベルか IO．SYS, MSDOS．SYS がこの順番に並んでいなければなりません．さもなければ，'No system file' と表示してホールトしてしまいます．

IO．SYS, MSDOS．SYS が存在すればこれらのプログラムは**図 1･3** のように連続して記録されているので，0060：0000H～のメモリに一気に読み込みます．完全に読み込めたら IO．SYS の先頭アドレス 0060：0000H へファージャンプします．そこには MS-DOS システムイニシャライザへのジャンプ命令が記録されています．

1・3・3　MS-DOS イニシャライザ

IO．SYS と MSDOS．SYS はいったん**図 1･5（a）**のようにメモリに読み込まれますが，IO．SYS の先頭アドレスに制御が渡されたとき JMP 命令でシステムイニシャライザへ飛びます．システムイニシャライザはひととおり IO．SYS のイニシャライズをしたあとクレジットタイトルを表示し，CONFIG. SYS の解析を行うルーチンが RAM 上限に**図 1・5（b）**のように転送され実行されます．

CONFIG. SYS 解析ルーチンでは，MSDOS．SYS をシステムイニシャライザをつぶすように移動します（**図 1･5（b）**）．つぎに MSDOS．SYS のイニシャライズを実行し，カレント PSP をセットし CONFIG. SYS ファイルがあれば，CONFIG. SYS 解析ルーチンのすぐ前にロードします（**図 1・5（c）**）．

ここまでの処理がすむと，システムイニシャライザは内部バッファや内部 FCB エリアを CONFIG．SYS の指定どおりに確保します（**図 1・5(d)**）．

内部 FCB を FILES が 5 以下のときに MSDOS．SYS のうしろに確保しないのは，IO．SYS の先頭でデフォルトの内部 FCB（CON，PRN，AUX，ユーザー用などの FCB）が 5 つ用意されているからです．

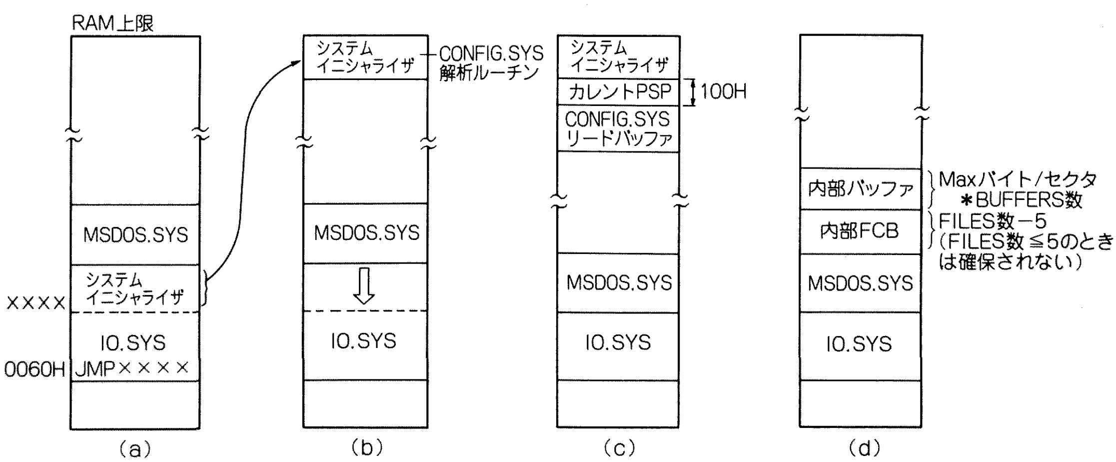


図 1・5　システムイニシャライザ

● CONFIG . SYS の解析ルーチン

CONFIG. SYS は同時にオープンできるファイルの数などを指定することができます. CONFIG. SYS で指定できるものは以下のものです. ( ) 内はデフォルト値です.

BUFFERS………ディスクバッファの数 (2)
BREAK …………ファンクションコールのたびごとに ^C チェックをする (OFF)
SHELL …………コマンドプロセッサのパス名指定 (¥COMMAND . COM)
DEVICE ………ユーザー・デバイスドライバの設定 (デフォルトなし)
FILES …………同時にオープンできるファイル数 (5)

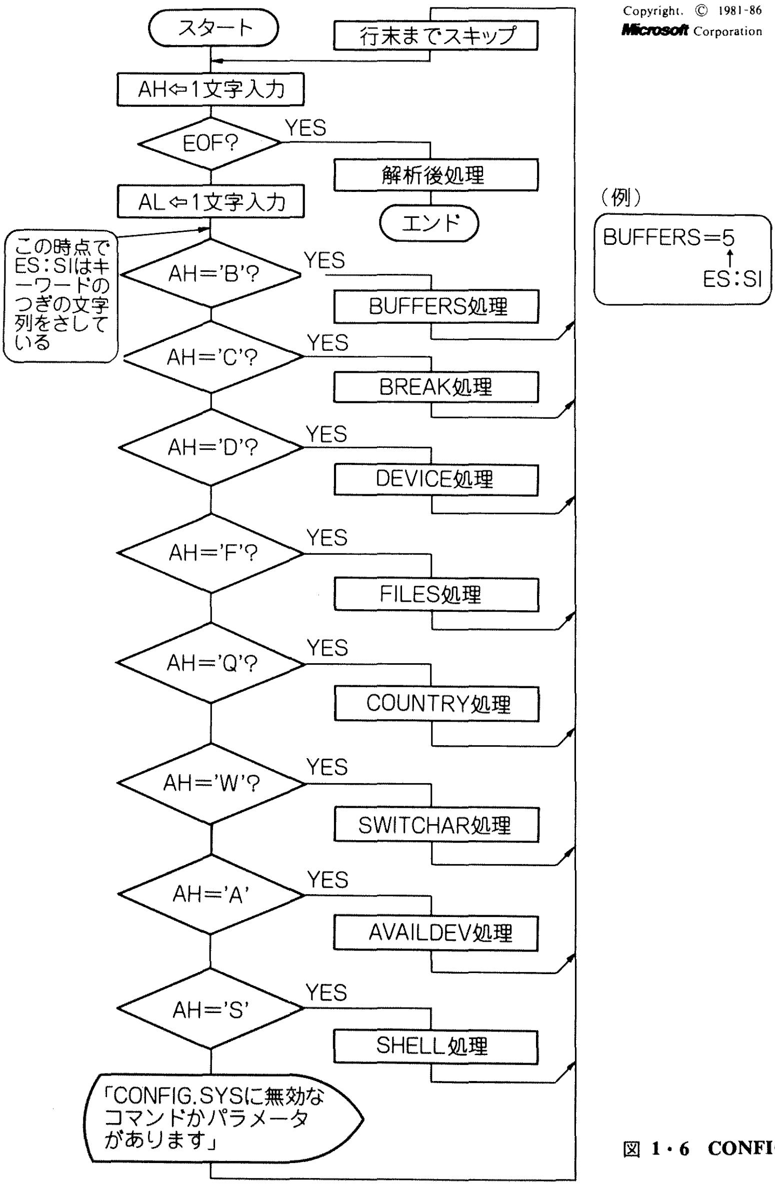


図 1・6 CONFIG. SYS キーワードの処理

CONFIG. SYS

```
DEVICE=RAMDISK21.COM 8 3
SHELL = A:¥COMMAND.COM /P
BUFFERS = 10
FILES = 6
```

```
-D0,4F
9F63:0000  44 52 41 4D 44 49 53 4B-32 31 2E 43 4F 4D 00 38  DRAMDISK21.COM.8
9F63:0010  20 33 0D 0A 53 41 3A 5C-43 4F 4D 4D 41 4E 44 2E   3..SA:¥COMMAND.
9F63:0020  43 4F 4D 00 2F 50 0D 0A-42 31 30 00 0A 46 36 00  COM./P..B10..F6.
9F63:0030  0A 2F 50 0D 0A 42 55 46-46 45 52 53 20 3D 20 31  ./P..BUFFERS = 1
9F63:0040  30 0D 0A 46 49 4C 45 53-20 3D 20 36 0D 0A 1A 00  0..FILES = 6....
```

SHELL　DEVICE　BUFFERS　FILES

内部コードに変換する前の文字列の残り

図 1・7　内部型式に変換された CONFIG. SYS の例

COUNTRY ……国番号の指定（81＝日本）

SWITCHAR ……コマンドのパラメータスイッチ指定（／）

AVAILDEV …… CON などのデバイス名を許すか（許す）

このうち SWITCHAR と AVAILDEV は，公式には発表されていませんが機能は内蔵されています（注：MS-DOS バージョン 3.0 以上では SWITCHAR と AVAILDEV はなくなっている）.

これらのキーワードはメモリに読み込まれたらまず内部コードに変換されます．内部コードは英字 1 文字で表わされ，キーワードとの対応は以下のようになっています．

キーワード	内部コード
BUFFERS	B
BREAK	C
SHELL	S
DEVICE	D
FILES	F
SWITCHAR	W
AVAILDEV	A
COUNTRY	Q

内部コードに変換されたあとは**図 1・6** のように解析を進めていきます．**図 1・7** に実際の CONFIG. SYS ファイルの内容と内部コードに変換した結果の対応の例を示しておきます．これらの内容は，システムが立ち上がったあとすぐは RAM 上限付近にゴミとなって残っています．

それぞれの処理内容について説明します．

① BUFFERS 処理

キーワードのつぎの文字列をバイナリ数字に変換し，数字以外の文字列か 100 以上なら「CONFIG. SYS に無効なコマンドかパラメータがあります」と表示しつぎのライン解析へ．100 未満だったらバッファ数へ保存．0 だったら単に無視してつぎのライン解析へ．

② BREAK 処理

キーワードのつぎの文字列が ON ならファンクションコール AH＝33H，DL＝1 (Break on) を実行しつぎのライン解析へ．ON 以外なら単に無視してつぎのライン解析へ(何もしなければデフォールトで BREAK＝OFF になっている)．すなわち，ON 以外の文字は OFF とみなされます．

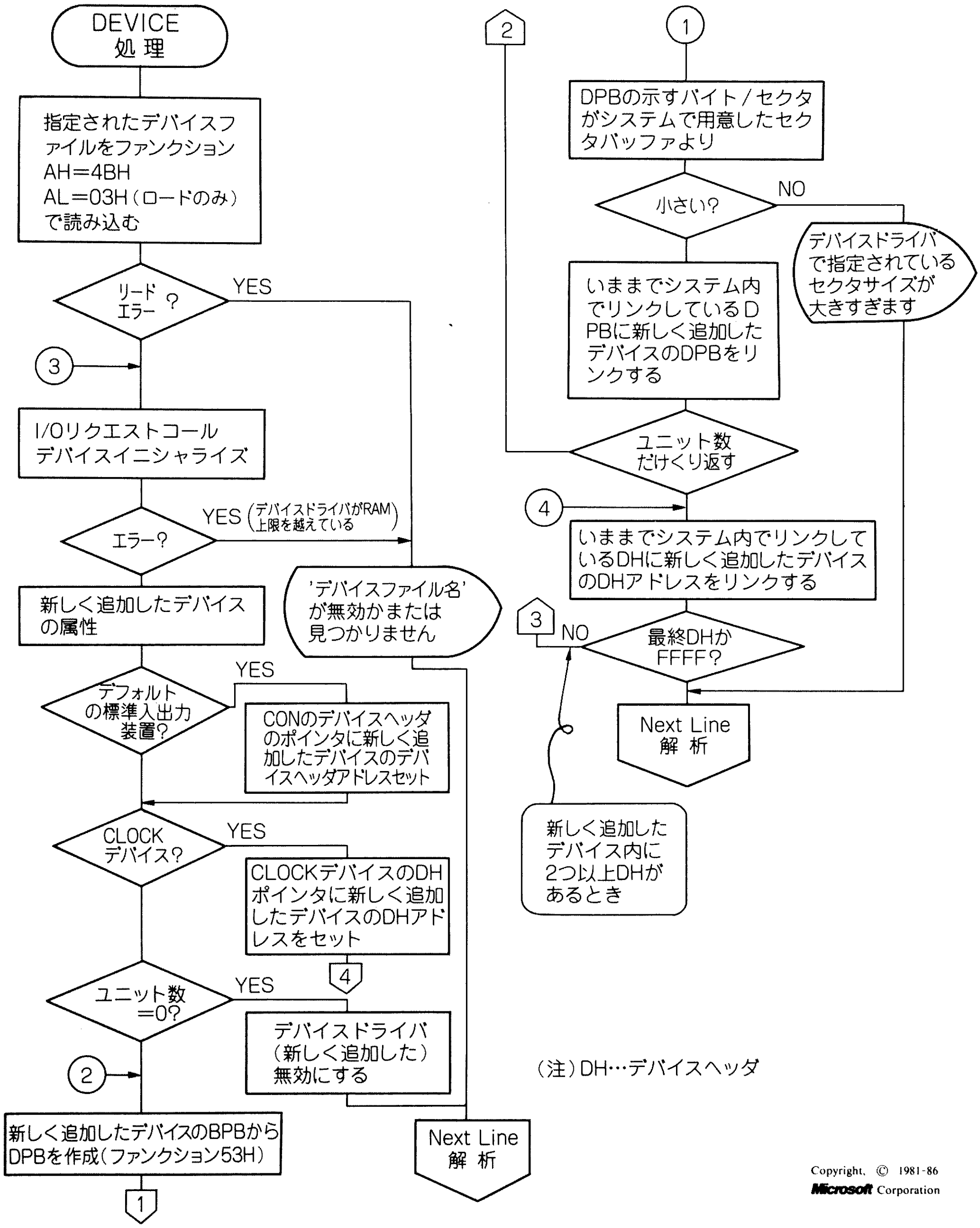


図 1・8 DEVICE 処理フローチャート

③ DEVICE 処理

大まかな流れを**図1・8**に示します．デバイスは MSDOS．SYS のあとのメモリに読み込まれイニシャライズされて，BPB・DPB のリンクに組み込まれます．I/O リクエストコールは MSDOS．SYS 内部ルーチンを使って行われますが，このイニシャライズ時，引数を格納するコマンドパケットには**図 1・9** のような内容が格納されています．コマンドパケットの（＋12～＋15H）の内容は CONFIG.

SYSファイルをメモリに読み込んだときのデバイス名部分をさしていますので，初期化時にたとえばRAMディスクならRAMディスクの大きさなどの引数を与えることができます．

④ **FILES処理**

キーワードのつぎの文字列をデシマル→バイナリ変換します．0なら単に無視しつぎの行解析へ．数字以外の文字または100以上なら「CONFIG. SYSに無効なコマンドかパラメータがあります」と表示し次行解析へ．

100以下なら同時にオープンできるファイル数に格納しておきます．

⑤ **COUNTRY処理**

キーワードのつぎの文字列をデシマル→バイナリ変換します．数字以外の文字があれば「CONFIG. SYSに無効なコマンドかパラメータがあります」と表示し次行解析へ．0か256以上なら単に無視して次行解析へ．

ファンクションコールAH＝38H，DX＝FFFFH(国別情報セット)を使って国番号をセットします．もしエラーなら「カントリーコードが無効です」と表示して次行解析へ．

⑥ **SWITCHAR処理**

キーワードの引数1文字をパラメータスイッチの区切り記号として登録します．

ファンクションコールAH＝37H，AL＝01H(パラメータスイッチ登録)を使っています．このファンクションコールも未公開ですが，MS-DOSバージョン3.0になってもAL＝0～2までは有効のようです．ただし，CONFIG. SYSからSWITCHARは削除されています．ちなみにAL＝0で呼び出すと現在のパラメータスイッチをDLに格納して戻ります．デフォールトは／(2FH)です．AL＝2，3についてはつぎのAVAILDEVで説明します．

パラメータスイッチとは

```
A>DIR  /W
```

などとしたときの／ (2FH) のことです．

⑦ **AVAILDEV処理**

AVAILDEV＝Fと指定するとCON，PRN，AUX，NULなどのデバイス名の機能をストップさせます．F以外は何もしません．

+0	コマンドパケットレコード長（この場合16H）バイト数	この部分の構造と大きさはコマンドコードにかかわらず同じ「リクエストヘッダ」という．
+1	論理装置コード（この場合0）	
+2	コマンドコード（0…イニシャライズ）	
+3,4	リザルトステータス	
+5～+C	リザーブ（システム）	
+D	ユニット数…デバイスドライバ側でセット	コマンドコード＝0 イニシャライズのときの構造．コマンドコードによって可変（他のコマンドについては第2章3節I/Oリクエストコール参照）
+E～+11	ブレークアドレス（デバイスドライバのENDアドレス）…〃	
+12～+15	BPBの配列へのポインタ…デバイスドライブ側でセット CONFIG. SYSファイル内のDEVICE＝の引数へのポインタ DEVICE＝RAMDISK 256 ↑ ここをさしている	

図1.9 CONFIG. SYSでDEVICEイニシャライズ時のコマンドパケットの内容

BX	内　容
+0, 1	環境のセグメント (0000)
+2～+5	PSP の 80H～FFH に与える内容へのベクタ
+6～+9	PSP の第 1FCB(5CH～)に与えるデータへのベクタ
+A～+D	同上第 2FCB (6CH～)

図 1・10 SHELL によって起動されるコマンドプロセッサに与えるパラメータブロック

ファンクションコール AH=37H, AL=3, DL=0 (デバイス名凍結) を使っています. 解除 (デバイス名を使用可能)にしたいときは DL=1 にします. AL=2 とすれば現在の状態を DL に格納して戻ります.

「デバイス名の機能をストップさせる」とはたとえば

A>DIR　>PRN

としたときに本来ならプリンタに出力されるところを PRN というファイル名が作られ, その中に書き込まれてしまうことを意味します.

MS-DOS バージョン 3.0 以上では CONFIG. SYS から AVAILDEV は削除されています. またファンクションコール AH=37H, AL=3 は何もしないように変わっています.

⑧ SHELL 処理

SHELL=のすぐうしろの文字列を, 後処理でコマンドプロセッサを起動するときのコマンドプロセッサファイル名に転送します. デフォールトは¥COMMAND . COM となっています.

そのあとの文字列を, コマンドプロセッサを起動するときの PSP の 80H～FFH に与えるバッファに格納します.

後処理の最後(**図 1・6** 参照)でコマンドプロセッサを起動しますが, ファンクションコール AH=4BH, AL=0 (プログラムロード & 実行) を使っています. このとき実行するプログラムにパラメータブロックとして**図 1・10** のような内容のテーブルを用意してこのテーブルの先頭アドレスを DS : BX に格納しますが, 上述したバッファへのポインタがブロック内のベクタとして格納されます.

CONFIG . SYS の SHELL で指定されたコマンドプロセッサの環境のセグメントアドレスは 0000H が指定されますので, 最初のコマンドプロセッサの環境のセグメントアドレス(PSP の 2CH, 2DH の内容) は 0000H となっています.

環境はコマンドプロセッサによって管理されるものなのです.

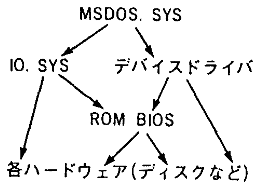
CONFIG . SYS の解析が終了したところで, FILES の値と BUFFERS の値にしたがってシステム・ワークバッファが確保されます (**図 1・5** の **(d)**). IO . SYS の内部に最初から 5 つ, 内部 FCB が作られています (そのうち 0 番目が AUX, 1 番目が CON, 2 番目が PRN, 残りの 2 つはユーザー用) ので FILES 数が 5 を越えた分だけ MSDOS . SYS のすぐうしろに (接続されているディスクの最大バイト/セクタ数) * BUFFERS 数 (バイト) のバッファエリアが確保されます (**図 1・5(d)**).

1・3・4 デバイスを中心にみた MS-DOS 概要

FCB, BPB, DPB など略語が多く出てきますが, これらの MS-DOS 上での位置を**図 1・11** に示し

ます．コンソールでユーザーが直接接するのは外部コマンドまたは COMMAND．COM です．各プロセスにはかならず 1 つの PSP があり，システムとプロセスのインタフェースをとりもちます．プロセスは INT 21H ファンクションコールでシステムに命令を発することができます．

システムは



という階層構造をしており，IO．SYS には 5 インチフロッピーや 8 インチフロッピー，ハードディスクなどのデバイスドライバが内蔵されています．必要ならばデバイスドライバとして新しく RAM ディスクなどのドライバを増設できます．IO．SYS やデバイスドライバは，そのエントリアドレスやデバイス名を記録したデバイスヘッダと呼ばれるテーブルを持っています．

デバイスドライバの実行開始アドレスを割り込みエントリといいます．

MSDOS．SYS は割り込みエントリを CALL してデバイスドイラバや IO．SYS へ命令を送り，データはストラテジエントリを CALL して，コマンドパケットアドレスをデバイスドライバへ通知

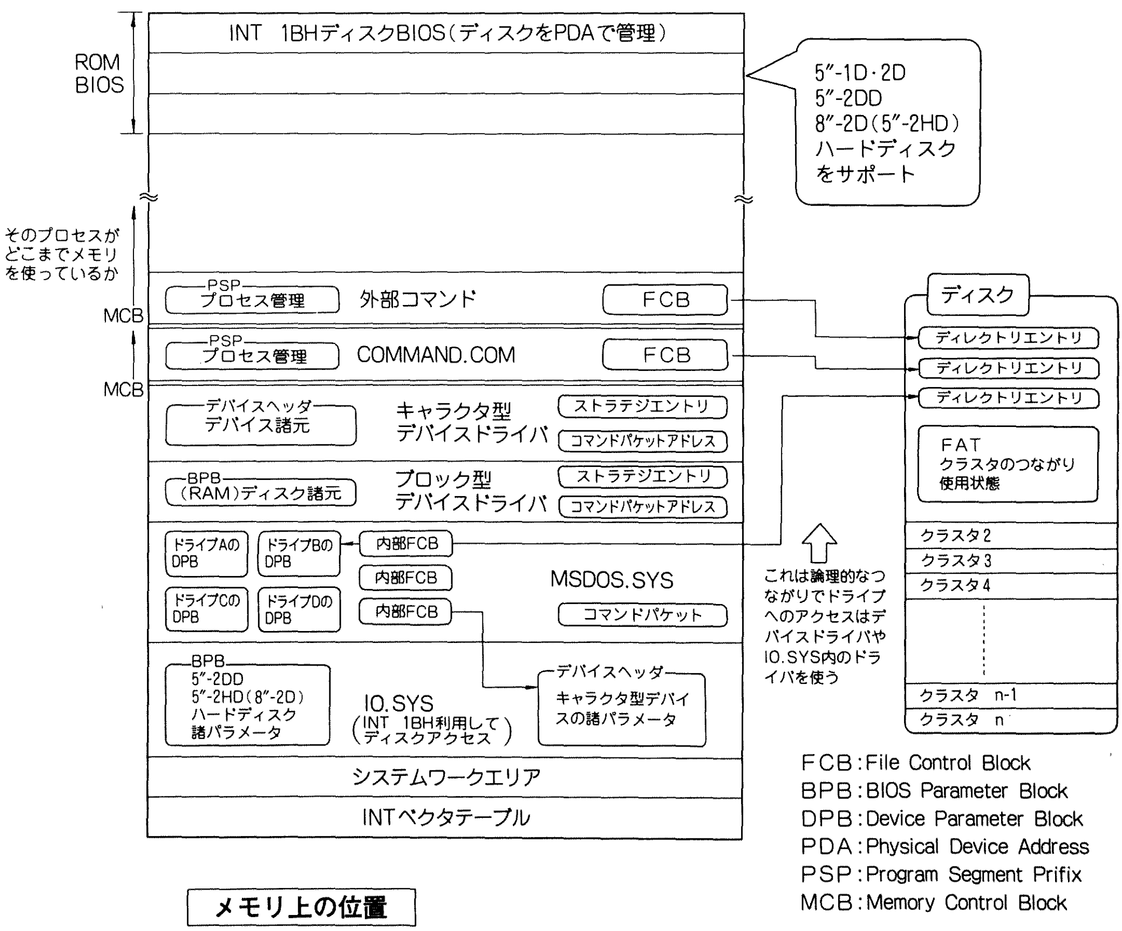


図 1・11　デバイス関係システムのメモリ上の配置

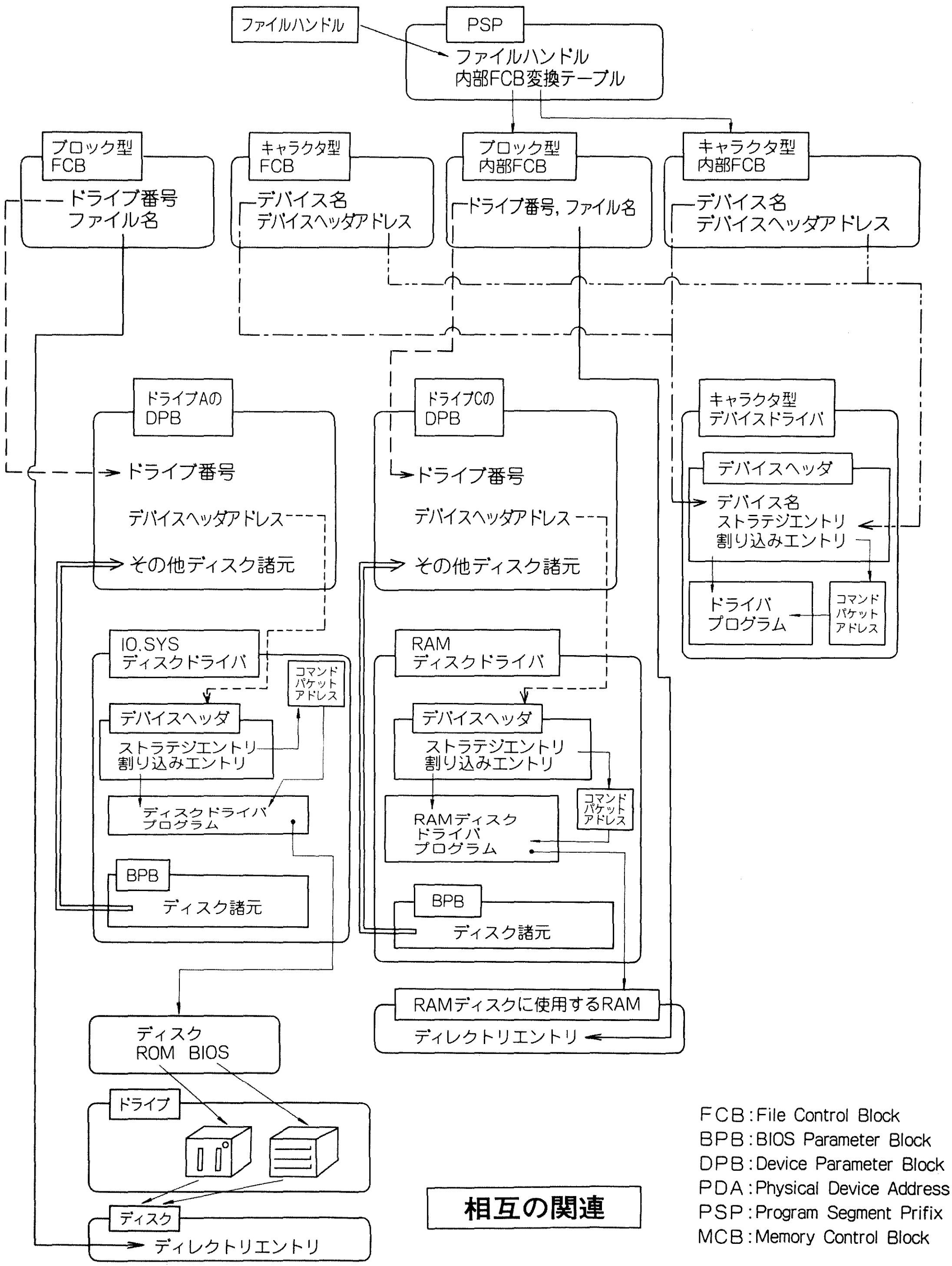


図 1・12　デバイス関係のシステム構成

し，コマンドパケットを通してやりとりします．

IO.SYS やデバイスドライバは，ディスクすなわちブロック型のデバイスに関してはセクタ長が何バイト，**FAT** の数がいくつかなどのディスク諸元を記録した **BPB** というテーブルを持っていま

す.

MSDOS.SYSは各ドライブ（A：, B：, C：, ...）がどういうBPB諸元を持っているかチェックし, 各ドライブごとにDPBというディスクの諸元をまとめたテーブルをそのBPBから作成します. したがって, DPBはドライブの数だけ存在することになり, BPBはそのデバイスドライバまたはIO.SYSがサポートするディスクの種類の数だけ存在することになります.

IO.SYSはディスクドライブをアクセスするためにPC-9800シリーズ固有のROM BIOSコールINT 1BHを使用しています. INT 1BHはPDAという形でディスクドライブを管理しています. PDAは各ディスクドライブの種類に応じて番号をつけ, 同じ種類のドライブが複数接続されていれば順に番号をつけたものです. コマンドパケットで指定する論理装置コードはこのPDAに対応しています.

FCBは内部（MSDOS.SYS内にとられるもの）と外部（通常のFCB）の2種類ありますが, いずれもディスク上のファイルまたはキャラクタ型デバイスをフアイル名, デバイス名で指定してデバイスと情報をやりとりするための情報が記録されます.

キャラクタ型デバイスはデバイス名をFCBで指定してアクセスしますが, ブロック型デバイスはディレクトリエントリ名をFCBで指定してアクセスします. とくにブロック型デバイスではディレクトリエントリに記録されているファイル情報そのものをFCBにコピーしてきて書き込みなどで変化したファイル情報をふたたびディレクトリエントリに書きもどしています(**図1•12**参照).

1・3・5 COMMAND.COMのイニシャライズ

IO.SYSとMSDOS.SYSのイニシャライズが終了すると, COMMAND.COMが起動されます. COMMAND.COMは大きく分けてつぎの3つの部分よりなります.

① イニシャライズ部

② 常駐部

③ 非常駐部

これらのメモリ上の配置は**図1・13**のようになっていて, イニシャライズがすめば①イニシャライズ部は外部コマンドエリアとして解放されます.

COMMAND.COMが起動されると, 先頭アドレスにあるイニシャライズ部へのJMP命令が実行され, 主導権がCOMMAND.COMイニシャライズ部へ渡ります.

イニシャライズ部では, 常駐部の中にある環境のセグメントアドレスの初期化, パイプ処理等の初期化をし, COMMAND.COMに与えられた文字列を実行します. COMMAND.COMに与えられる文字列としては以下の3つのパラメータがあります.

/c……コマンド実行

/d……イニシャライズ後, 日付・時刻の入力をもとめない.

/p……パーマネント属性

それぞれの文字列が検出されたらおのおのに対応するフラグを立て解析を継続します. /cだけはコマンドの実行処理のほうへ移行します. /cが指定されると/pフラグは非パーマネント, /dフラグは非入力とされます.

上記以外の文字列がコマンドラインにあると, それはCOMMAND.COMの再入力パス名とみなされます. このことはCONFIG.SYSの処理の中のSHELLコマンドと関係しています. CONFIG.SYSではSHELL=¥COMMAND.COMのように書いて, SHELL=のあとに書かれた文字列を

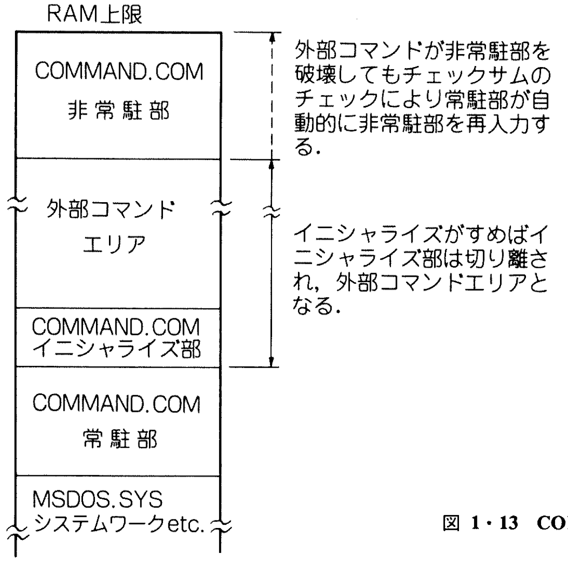


図 1・13 COMMAND.COM メモリ配置

そのままCOMMAND.COMへ渡しますので，COMMAND.COMのほうではCOMMAND.COMのうしろに書かれたコマンドラインでパラメータ以外の文字列をCOMMAND.COMの再入力パス名とみなすのです.

イニシャライズ部ではいったんこの文字列をASCII文字列にしてオープンしてみます. オープンできればよいがオープンできなければCOMMAND.COMをそのパス名にセットしてふたたびオープンしてみます. それでもオープンできなければ「COMMAND.COMが見つかりません」と表示します. うまくオープンできれば，環境にCOMSPEC='オープンできたパス名'をセットしてコマンドライン解析を継続します.

コマンドライン解析が終了すれば，COMMAND.COMのPSPの初期化に移ります.

PSPのオフセットは

A～DH	終了アドレスに常駐部のエントリアドレス
E～11H	コントロールCの処理アドレス（これも常駐部の中にある）
16, 17H	常駐部のセグメント

をセットします.

プリンタスプーラ用の処理アドレスをINT 2EHベクタにセットします. COMMAND.COMの常駐部のワークエリアにも(16, 17H)でセットしたCOMMAND.COMのPSPセグメントを保存し，PSPの(18, 19H)の標準ファイルハンドラ番号やデフォールトのFCBアドレスとしてPSPのセグメントとオフセット5CH, 6CHをDTAアドレスとしてオフセット80Hをセットします.

ここまできたらCOMMAND.COMの常駐部の大きさのみにメモリブロックの割り当てを変更し，イニシャライズ部を切り離します. 先きほどの再入力用のパス名を使って非常駐部をメモリの最上位アドレスに読み込み, 2バイトずつ足していく方法によってチェックサムをとり, 常駐部のワークエリアに記録しておきます. チェックサムはあとで外部コマンドを実行したあとで非常駐部が破壊されたかどうかのチェックに用います. ここまでうまくいけば

Command バージョン 2. 11

と表示します. コマンドライン解析のときにセットした/dフラグを見て，日付・時刻の入力を求めないときはそのまま常駐部のエントリへジャンプし，入力を求めるときはカレントドライブの

AUTOEXEC.BAT がオープンできるか試してみます．オープンできなければ非常駐部にある日付·時刻入力ルーチン（DATE，TIME コマンドエントリ）をファーコールし常駐部エントリへジャンプします．AUTOEXEC.BAT がオープンできれば，バッチ処理用の 48 バイト（3 パラグラフ）のワークエリアを確保し，バッチ処理フラグを ON にして常駐部エントリへジャンプします．

バッチ処理用ワークバッファにはバッチファイルのコマンドラインに与えられた文字列引数へのポインタが記録されます．バッチ処理自体はバッチ処理中かどうかのフラグを見て常駐部の方で行われますので，COMMAND.COM のイニシャライズ部の処理はここまで，ということになります．

COMMAND.COM のイニシャライズがすむと，MS-DOS を動かしてゆくためにはイニシャライズルーチン自体は無用のものになってしまいます．そこで，イニシャライズルーチンの最後でイニシャライズルーチンが占めているメモリを解放してしまう必要があります（このメモリの解放にはファンクションコール 49H が使用される）．イニシャライズルーチンは COMMAND.COM の常駐部のすぐうしろに位置しているので，解放されたメモリは外部コマンドエリアとして使用されます．COMMAND.COM のすぐうしろにはバッチ処理用ワークエリアと外部コマンド環境エリアがあります．

COMMAND.COM 非常駐部読み込みについて

COMMAND.COM の非常駐部を読み込むときに，よくメモリに読み込まれた COMMAND.COM の非常駐部部分をメモリの最上位アドレスに転送するといわれていますが，ほんとうは

① ファンクションコール番号 3DH の AL=0（リードオンリ・オープン）によってパス名で COMMAND.COM をオープンする．

② ファンクションコール番号 42H の AL=0（先頭から CX：DX バイト目へファイルポインタ移動）を用いて非常駐部トップへファイルポインタを移動する（**図 1・14**）．

③ ファンクションコール番号 3FH（ファイルデバイスの読み出し）を使って非常駐部をメモリへ読み出す（CX ←非常駐部の大きさ）．

という手順でディスクから直接最上位メモリへ読み込みます．ファイルデバイスの読み出しのエラーチェックとして，キャリーフラグがセットされて戻ればエラーですが，その上に実際に読み込まれたバイト数が AX レジスタに返ってきますので，非常駐部の大きさをセットした CX レジスタと比較して，AX≠CX であればエラーとします（全部読めたかのどうかのチェック）．

この非常駐部再入力ルーチンは常駐部にあります．

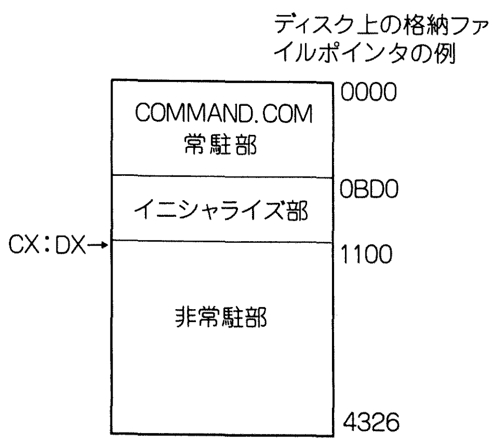


図 1・14 ディスク上の **COMMAND.COM**

COMMAND.COM のイニシャライズ部と非常駐部のファイルポインタの求め方

図 1・14 に示したディスク上のファイルポインタの値は，バージョンによって異なりますが簡単に見付けることができます．まず DEBUG. COM で

A>DEBUG COMMAND.COM

として COMMAND.COM を読み込みます．R コマンドでレジスタを表示させると読み込んだバイト数が CX レジスタに入っています．図 1・14 の 4326H の値のことです．つぎに，このままではオフセット 100H から読み込まれているので，DS と CS レジスタの値を

現在の値＋10H

に変更します．DS レジスタは RDS，CS レジスタは RCS と入力すると新しい値を求めてきますので，変更値を入力してください．

つぎに，ディスアセンブルコマンド U で U0 としますと

JMP 0B00

…

と表示されます．このジャンプ先のアドレスがイニシャライズ部の TOP です．

非常駐部の TOP はサーチコマンド S を使って

S0 3DD0 B4 0E CD 21(注)

↑この値は COMMAND.COM を読み込んだときの CX レジスタの値

としますと

56C2：1100

などと表示されますので，このオフセットの方の値となります．

以上の操作の例を図 1・15 に示します．

```
A>
A>debug command.com
-r                           COMMAND. COM 全体の大きさ
AX=0000  BX=0000  [CX=4326]  DX=0000  SP=FFFE  BP=0000  SI=0000  DI=0000
DS=12A8  ES=12A8  SS=12A8  CS=12A8  IP=0100   NV UP DI PL NZ NA PO NC
12A8:0100 E9CD0B        JMP     0CD0
-rds
DS 12A8
:12b8 ……DS レジスタに現在の値 12A8H+10H=12B8H をセット
-rcs
CS 12A8
:12b8 ……CS レジスタに現在の値 12A8H+10H=12B8H をセット
-u0
12B8:0000 E9CD0B        JMP     [0BD0] ……イニシャライズ部 TOP
12B8:0003 0000          ADD     [BX+SI],AL
12B8:0005 0000          ADD     [BX+SI],AL
12B8:0007 0000          ADD     [BX+SI],AL
12B8:0009 0000          ADD     [BX+SI],AL
12B8:000B 0000          ADD     [BX+SI],AL
12B8:000D 0000          ADD     [BX+SI],AL
12B8:000F 0000          ADD     [BX+SI],AL
12B8:0011 0000          ADD     [BX+SI],AL
12B8:0013 0000          ADD     [BX+SI],AL
12B8:0015 0000          ADD     [BX+SI],AL
12B8:0017 0000          ADD     [BX+SI],AL
12B8:0019 0000          ADD     [BX+SI],AL
12B8:001B 0000          ADD     [BX+SI],AL
12B8:001D 0000          ADD     [BX+SI],AL
12B8:001F 0000          ADD     [BX+SI],AL
-s0 4326 b4 0e cd 21
12B8:[1100] ……非常駐部 TOP
-
-q
```

図 1・15　COMMAND.COM のイニシャライズ部 非常駐部のファイルポインタの見つけ方

注）　B4　0E　CD　21
MOV AH, 0E
INT 21H

のことで，ディスクのイニシャライズです．非常駐部はかならずこのコマンドから始まっていることを利用します．

1・3・6　システムの立ち上げのまとめ

1・3・1〜1・3・5で，リセットボタンを押してから COMMAND．COM が起動するまでを見てきましたが，ここで手短かに**図 1・16** にまとめておきます．

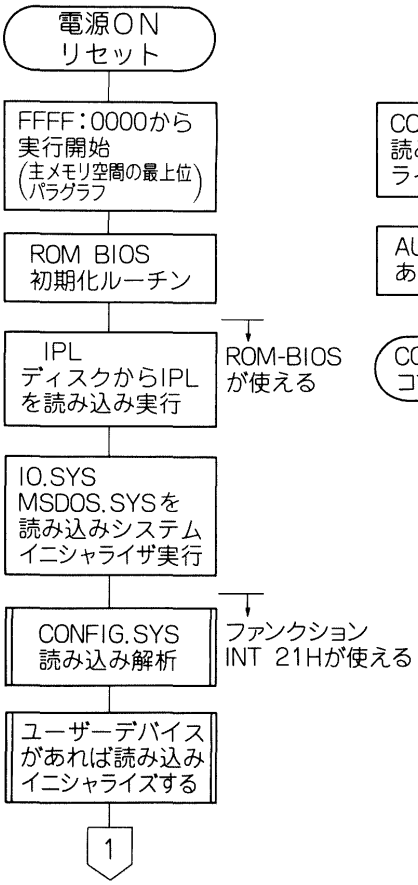


図 1・16　リセットからコマンド入力まで（まとめ）

電源 ON で，FFFF：0000 から実行開始というのは，8086 CPU 固有の機能ですから，他の OS にも共通しています．また，ROM BIOS 初期化と IPL は PC-9801 に固有のものです．

初期化の順番は ROM-BIOS，IO．SYS，MSDOS．SYS，ユーザーデバイス，COMMAND．COM の順番ですから，CONFIG．SYS で登録するユーザーデバイスでは INT 21H コールは使えます．ユーザーデバイスの初期化の時点では，ファンクション 9 番の文字列表示などで文字を表示すること

もできます．また，AUTOEXEC. BAT では COMMAND. COM の内部コマンドや，外部コマンドも使えます．

CONFIG. SYS の解読処理は，MSDOS. SYS から実行され，AUTOEXEC. BAT は COMMAND. COM から自動実行されます．そのため，CONFIG . SYS はシステムに関するパラメータやデバイスを登録する機能がありますが，AUTOEXEC . BAT はコマンドシェルの上の単なるバッチファイルとしての性格しかありません．ですから，AUTOEXEC. BAT で実行されるコマンドはコマンドシェルの内部コマンドか，外部コマンド，またはバッチファイルだけです．ただし，バッチファイルを AUTOEXEC. BAT の中で実行すると，そのバッチファイルを実行終了した時点で AUTOEXEC. BAT の実行も終了してしまいます．

ここで，ちょっと便利なバッチファイルについてひとこと説明しておきましょう．

バッチファイルの中で，バッチファイルを実行する方法は，バッチファイルの中で

COMMAND/C <実行したいバッチファイル名>

というように，COMMAND. COM を外部コマンドとして，/C オプションを使って他のバッチファイルを実行させれば，他のバッチファイルの実行が終わったところで，もとのバッチファイル処理が戻ってきます．

バッチ処理はディスクの上にワークファイルを作るので，処理が遅くなります．ところが，バッチ用コマンド FOR は COMMAND. COM のワークエリアの中にバッファを持っていますので，ディスクにワークファイルを作ることなくスムーズに実行されます．もちろん，バッチファイルも RAM ディスク上で実行すればスムーズに実行されます．ただし，FOR 用のワークエリアは 2 行～3 行分しかないので，あまり長い FOR 文はバッファがオーバフローして実行できません．

バッチコマンド FOR の使いかたは，たとえば，アセンブル用のコマンドを RAM ディスク C：に COPY するようなときは

FOR %V IN (MASM LINK EXE2BIN) DO COPY %V. EXE C :

というように，COPY したいコマンドを IN ( ) の中に並べて，DO のあとに COPY 命令を書きます．このとき FOR のすぐあとに書いた変数（この場合，変数は%変数名という形で書く）%V に，IN ( ) の中の文字列が順番に代入されて，DO 以降の実行文の中の%V が，その文字列と置き換わり，COPY 命令が実行されます．

Ver 2. xx では，FOR の処理の中にちょっとしたバグがありましたが，Ver 3. xx ではそのバグもフィックス[注] され，IN ( ) の中の文字列にワイルドカードが使えます．また，FOR 文の文字列がバッファオーバーしたときにごみの文字列が実行文にくっついて実行されるということもなくなっています（このバグについては，Ver 2. xx でもフィックスされているものもあります）．

[注] フィックスとはバグが発生しないように訂正すること．

第2章

IO.SYSの仕組み

2・1 IO.SYSのワークエリア

IO.SYSはバージョンによってかなり違っていますが，セグメント60Hから始まっていることは共通しています．IO.SYSのワークエリアからは，接続されているフロッピードライブの種類や数，MSDOS.SYSのセグメントなどがわかります．

図 2・1にワークエリアの内容をまとめておきます．

オフセット	内　　容
(1)	システムイニシャライザへのジャンプ命令 (システムが立ちあがったあとではシステムイニシャライザは消されるので意味がない)
(2)	ここをファーコールすると日本語入力モードとなる
(3)	ここをファーコールすると1/2角文字入力となる (通常の文字)
(4)	文字列　KNJDIC␣␣SYS 漢字熟語辞書名．辞書はシステムディスクからしか読めない
(5)	ROM上のブートストラップローダ（5″-2DD用）アドレス
(6)	ROM上のブートストラップローダ（8″フロッピー用）アドレス
(7)	MSDOS.SYSのセグメント値
(8)	bit5… プリンタ接続
(9)	PDAテーブル．順にドライブA，B，C…，PのPDA 70H台…5″-2DD　80H台…5″ハードディスク 90H台…8″　00H…接続ドライブなし（IO.SYSの管理の）
(10)	メディアディスクリプタ（何番目のBPBかということ）5インチフロッピー用
(11)	IO.SYSがサポートしているドライブの種類 bit3…8″-2D　bit4…5″-2DD
(12)	bit0… プリンタ存在
(13)	グラフィック文字（80～9FH，E0～FFH）を入力時 0：受けつける　1：受けつけない　ただし日本語入力モードは使えなくなる．CTRL+f·4
(14)	接続されている8″-2Dフロッピードライブ数
(15)	接続されている5″-2DDフロッピードライブ数
(16)	接続されているハードディスクドライブ数
(17)	bit0…ハードディスクドライブ1の存在 bit1…ハードディスクドライブ2の存在
(18)	システムディスクの種類　0…8″-2D　1…5″-2DD

図 2・1　IO.SYSのワークエリア（セグメント60H）

とくに利用価値が高いのは，ディスクに関するワークエリアです．ただし，IO.SYSが管理しているドライブだけで，デバイスドライバに関する情報は入っていません．まずドライブA～Pまでのドライブの種類に対応するPDAがあります．PDAというのは 'Physical Drive Address' (注) のことで，上位4ビットでドライブの種類を表わします．

ドライブの種類（上位4ビット）は

7……5"-2DDフロッピーディスク

8……5"ハードディスク

9……8"-2Dフロッピーディスク

となっています．下位4ビットのドライブ番号は最初のドライブが0，つぎが1，… と数えていきます．したがってPC-9801（V）F2をお持ちの人は

オフセットアドレス	内容
PDATOP+0H	70 …… ドライブA
+01H	71 …… ドライブB
+02H	00 …… ドライブC
+03H	00 …… ドライブD
…	… ……

となっているはずです．5"-2DDなので上位4ビットが7，ドライブが2台つながっているので，最初が下位4ビット0の70H，つぎが下位4ビットが1の71Hとなっているのです．(V) M2の人は70H，71Hの代わりに90H，91Hとなります．5"-2HDドライブは8"-2Dドライブとソフト的には同じ扱いになるためです．そのため5"-2DDを640kバイトタイプ，8"-2Dと5"-2HDを1Mバイトタイプと分類することもあります．基本ドライブ以外に何もつながないと，このように対応するドライブの項には00Hが入れられます．

デバイスドライバで追加したRAMディスクなどはこの表からはわかりませんが，ドライブAにどういうドライブが接続されているかなどは，この表を見れば知ることができます．また，各ディスクの接続されている台数や，システムディスクがどのタイプのものかを示す数字が入っています．

つまり，(13) には8インチ（5インチ2HD），(14) には5インチ2DD，(15) にはハードディスクの接続台数が入っていることになります．

2・2 デバイスヘッダ

"デバイスヘッダ" とはデバイス名などのデバイス管理に必要な情報をまとめた表のことです．IO.SYSはデバイス名CON，AUX，PRN，CLOCKの各デバイスとディスクデバイス(8"-2D[5"

注）PDAとはINT 1BH ROM BIOSで使用されているディスクドライブの分類コードで，MS-DOS標準のものではありません．INT 1BH ROM BIOSについては，2・10節のディスクデバイス中のコラム「INT 1BHディスクROM BIOS」を参照してください．

```
A>debug
-d646:fc,ff  ←MSDOS. SYS のワーク FC～FFH の中身を見る
0646:00FC     [HDTOP|60 00]  =セグメント 60H：オフセットHDTOP…デフォールトのデバイスヘッダTOP アドレス
-d[60:HDTOP]
0060:HDTOP     [72 26|60 00] 13 80 5C 27-67 27 43 4F 4E 20 20 20   r&`...¥'g'CON
0060:HDTOP+10  20 20 [84 26|60 00] 00 80-5C 27 6D 27 41 55 58 20     .&`...¥'m'AUX
0060:HDTOP+20  20 20 20 20 [96 26|60 00]-00 80 5C 27 73 27 50 52       .&`...¥'s'PR
0060:HDTOP+30  4E 20 20 20 20 20 [A8 26|60 00] 08 80 5C 27 79 27   N     (&`...¥'y'
0060:HDTOP+40  43 4C 4F 43 4B 20 20 20-[FF FF|FF FF] 00 20 5C 27   CLOCK   ..... ¥'
0060:HDTOP+50  7F 27 02 90 BC 27 BC 27-BC 27 B6 27 D0 27 EB 27   .'..<'<'<'6'P'k'
0060:HDTOP+60  BC 27 0A 28 1E 28 1E 28-4B 28 BC 27 BC 27 BC 27   <'.(.(.(K(<'<'<'
0060:HDTOP+70  BC 27 BC 27 B6 27 72 28-8E 28 BC 27 AF 28 C4 28   <'<'6'r(.(<'/(D(
-q

A>
```

図 **2・2**　実際のデバイスヘッダ

-2HD]，5"-2DD，ハードディスク)を管理するためのデバイスヘッダを内蔵しています．ディスクデバイスにはデバイス名の代わりにディスクの台数を記録するようになっています．各デバイスヘッダはつぎのデバイスヘッダのあるアドレスを記録しており，最後のデバイスヘッダのリンク情報のオフセット，セグメントアドレスとも FFFFH となっています．これをデバイスのリンク情報といいます．

それでは，実際のシステムの中のデバイスヘッダをのぞきながら解説を進めていくことにしましょう．デフォールトのデバイスヘッダ[注1]の先頭アドレスは，MSDOS . SYS のワークエリア（FCH～FFH[注2]）に記録されています．

MSDOS . SYS のセグメントは，IO . SYS のワークエリア (7) に記録されています (**図 2・1** 参照)．このセグメントの FC～FFH を DEBUG . COM でのぞいてみましょう．その様子を**図 2・2** に示します．MSDOS . SYS のセグメントアドレスとデバイスヘッダ TOP アドレスは，システムによって異なることがあります．この例ではデバイスヘッダ TOP アドレス HDTOP に対して

HDTOP ＋0～＋11H	CON 用デバイスヘッダ
HDTOP ＋12H～＋23H	AUX 用デバイスヘッダ
HDTOP ＋24H～＋35H	PRN 用デバイスヘッダ
HDTOP ＋36H～＋47H	CLOCK 用デバイスヘッダ
HDTOP ＋48H～＋59H	ディスクデバイス用

となっています．

ここでデバイスヘッダの内容を**図 2・3** に示します．**図 2・2** と対応させてみれば，最初のデバイスヘッダは CON デバイスヘッダで，リンク情報は 0060：HDTOP＋12H でつぎの AUX 用デバイスヘッダを指していることがわかります．CON は 3 文字ですが空白で補って 8 文字分の領域としています．属性は 8013H なので以下のようになります．

b15 ……キャラクタ型デバイス
b14 ……I/O リクエストの I/O コントロール使用不可
b13 ……(b15 がディスクのとき有効)
b4　……特殊デバイス
b3　……CLOCK デバイスでない
b2　……NUL デバイスでない

注1)　CONFIG . SYS で追加したデバイスも含めたデバイスヘッダの TOP へのポインタは，MSDOS . SYS ワークエリアの 107H～10AH にある．

注2)　Ver. 3. xx ではアドレスが異なる．

0～＋3	つぎのデバイスヘッダへのリンク情報（オフセット・セグメントの順）最後のデバイスヘッダはFFFFFFFFHとなっている
＋4,5	デバイス属性(注)
＋6,7	ストラテジエントリ
＋8,9	割り込みエントリ
＋A～11	デバイス名（8バイト） ディスクの場合はドライブ数で＋Aのみ

(注)

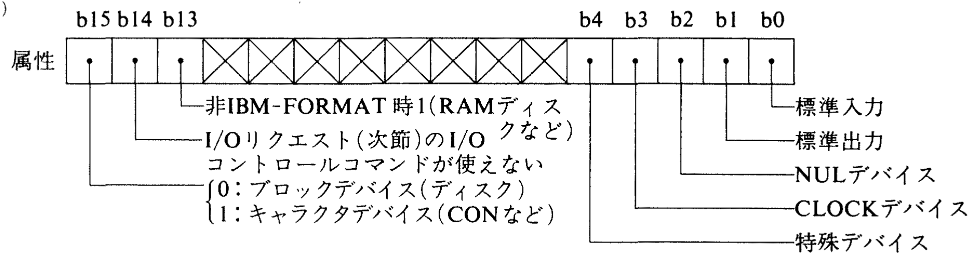


図 2.3 デバイスヘッダの内容

b1 ……標準出力デバイスである
b0 ……標準入力デバイスである

特殊デバイスというのは，INT 29HでALレジスタの中身を標準出力へ出力できるようになっているデバイスのことです．将来サポートされなくなるとのことですが，バージョン3.10まではまだ使えます．非IBM-FORMATとはディスクのフォーマットがIBM-PCとは異なっているということです．

CONのストラテジエントリと割り込みエントリの関係はつぎのようになります．

●ストラテジエントリの働き

ES：BXの値をIO.SYSのワークエリアに保存します．ES：BXは割り込みエントリで処理するのに必要なコマンドなどのデータ（引数）をまとめたテーブルへのポインタです（このテーブルのことを“コマンドパケット”といいます）．コマンドパケットについては38ページ参照．

●割り込みエントリの働き

ストラテジエントリで保存したコマンドパケットの内容を読み出し，指定されたコマンドを実行する．

ということです．

なぜこのようにコマンドテーブルを入力するエントリと実際に実行するエントリに分けているのかというと，MS-DOSが将来マルチタスクを意識しているからです．コマンドパケットアドレスをつぎつぎとためこんでいき，マルチタスクで実行したものからコマンドパケットアドレスのアドレスを捨てていくということができるのです（図2・4）．

コマンドパケットはコマンド実行に必要なデータが格納されたテーブルですが，コマンドパケット自体はI/Oリクエストコールを発したほうにあり，マルチタスク処理システムの中ではそのアドレス（セグメントとオフセットで4バイトですむ）を扱っています．

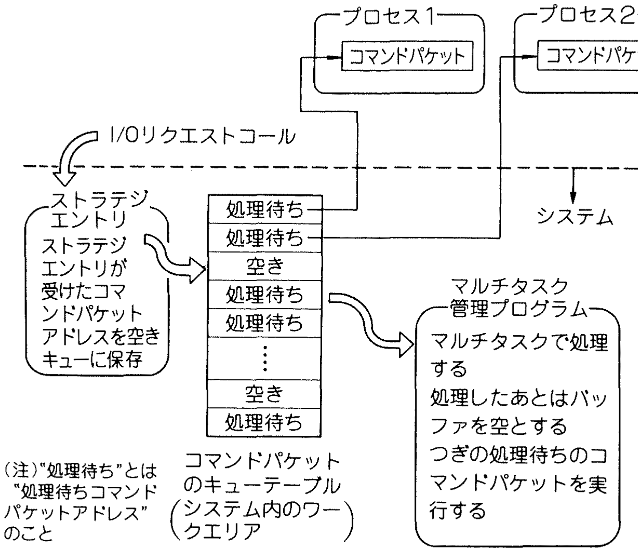


図 2・4　MS-DOS の将来の構想（想像図）

図の□1つ1つがコマンドパケットのアドレス格納エリアになっており，処理がすむごとに，つぎのコマンドパケットのアドレスをここから読み出して，つぎつぎにマルチタスクで処理することになります．

さて，**図 2・2** と**図 2・3** に戻って，各デバイスヘッダの関係を**図 2・5** に表わしてみます．

● CONFIG.SYS で追加されたデバイスドライバ

CONFIG.SYS の DEVICE で追加したデバイスドライバのデバイスヘッダは，いままで説明したデバイスヘッダのリンクのどこに追加されるのでしょう．

答えは CON のデバイスヘッダの前ということになります．デバイスヘッダ自体は IO.SYS に属しますが，その先頭アドレスを示すポインタは MSDOS.SYS のワークエリアにあります．Ver2.xx では MSDOS.SYS のセグメントのオフセット 107～10AH（**図 3・4** のシステムパラメータテーブルの＋17～1AH）に格納されています．

RAM ディスクや漢字熟語変換ドライバなどを CONFIG.SYS で追加しなければ，107～10AH の内容は IO.SYS で本来管理しているデバイスヘッダの先頭 CON デバイスヘッダを指しています（**図 2・5** 参照）．

しかし CONFIG.SYS の DEVICE でデバイスを追加したら，107～10AH の内容は追加したデバイスドライバのデバイスヘッダのアドレスとなり，追加されたデバイスヘッダを指すように変更されます（**図 2・6**）．

さきに**図 2・2** を出すときに用いた MSDOS.SYS 内のワークエリア FC～FFH は IO.SYS 本来のデバイスドライバ CON へのポインタで，CONFIG.SYS でデバイスが追加されても変更されません．

NUL デバイスは CONFIG.SYS でデバイスドライバが追加されたのち MSDOS.SYS により付加されます．NUL デバイスドライバの処理ルーチンは MSDOS.SYS 内にあります．したがってデバイスのリンクの状態は**図 2・6 (b)** のようになります．

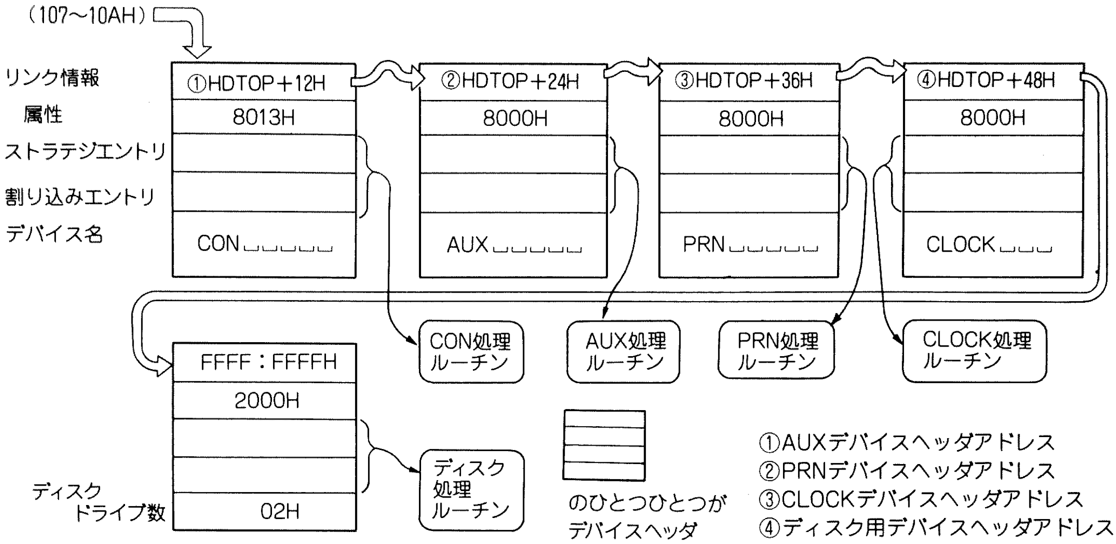


図 2・5　デバイスヘッダのリンク状態の例

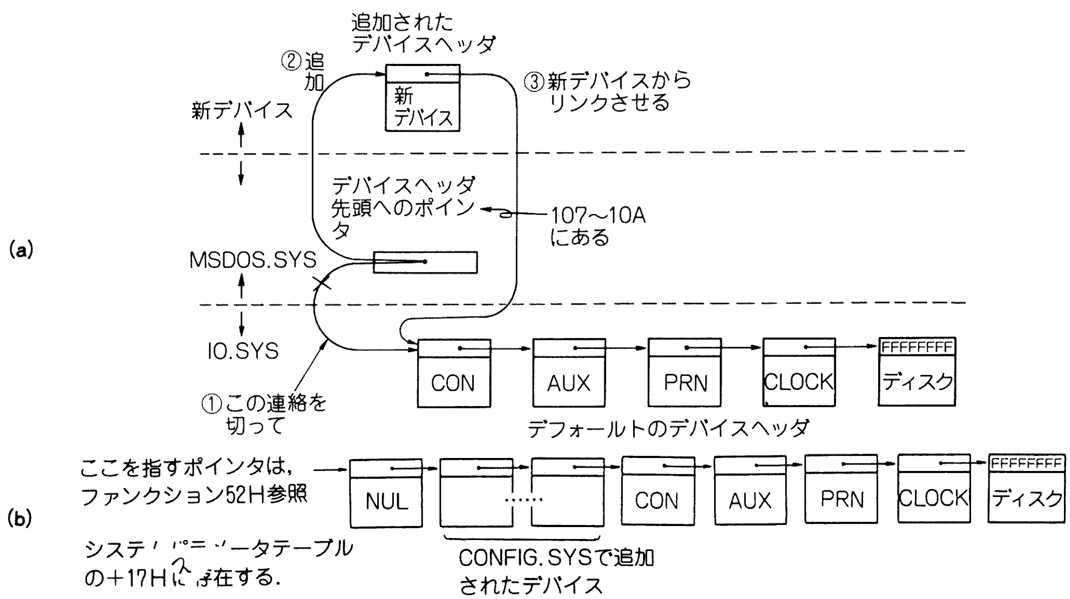


図 2・6　デバイスドライバが追加される様子

2・3　I/O リクエストコール

MSDOS.SYS から IO.SYS のデバイスドライバにコマンドを送る場合は，I/O リクエストコールを行います．I/O リクエストコールのエントリはデバイスヘッダの +6，7 と +8，9 に記録されています（**図 2・3**）．+6，7 のほうはストラテジエントリといって引数渡しのためのもので，実際のコマンド実行は +8，9 の割り込みエントリで行います．これらのアドレスはオフセットアドレスで，

コマンド番号	機　　能	キャラクタ型	ブロック型
0	イニシャライズ	○	○
1	ディスクの交換チェック	×	○
2	BPB テーブルのアドレスリクエスト	×	○
3	I/O コントロール機能（入力）	△	△
4	リードデータ	○	○
5	キュー非破壊入力	○	×
6	ステータスチェック（入力用）	○	×
7	入力バッファフラッシュ	○	×
8	ライトデータ	○	○
9	ライト ＆ ベリファイデータ	○	○
10	ステータスチェック（出力用）	○	×
11	出力バッファフラッシュ	○	×
12	I/O コントロール機能（出力）	△	△

×はなにもぜずにリターン
△はI/O コントロールをサポートしているときのみ

キャラクタ型：1バイト入出力するデバイスで，たとえば CON，PRN
ブロック型：ディスクのこ

図 2・7 I/O リクエストコールコマンド一覧

セグメントアドレスは対象としているデバイスヘッダがあるセグメントとなります．

具体的には，ES：BX にコマンド番号や引数を記録したテーブル（これをコマンドパケットという）を指させ，ストラテジエントリを FAR CALL し，引き続き割り込みエントリを FAR CALL します．

コマンド番号は 0～12 までであり，もしデバイスドライバを自作して CONFIG．SYS の DEVICE でシステムに組み込むなら，これらのコマンドを実行するプログラムを作ればよいことになります．ただし，**これらのコマンドは MS-DOS のファンクションコール 44H の“I/O コントロール”とは異なりますので，混同しないようにしてください**[注]．

I/O リクエストコールコマンドを**図 2・7** に示します．各コマンドの引数はコマンドパケットという領域をメモリ上に作成して，その先頭アドレスを渡すことによってデバイスドライバに伝えます．コマンドパケットは各コマンドによって引数の数や大きさが違うので可変長となっています．コマンドパケットの中で先頭 13 バイトはコマンドパケットの長さ（バイト数）やコマンド番号を記録するようになっており，**図 2・8** のリクエストヘッダのように各コマンド共通です．そのうしろにコマンド特有の可変長の引数エリアがきます．

リクエストヘッダ中“ステータス”は**図 2・9** のようになっており，コマンド実行の結果異常終了の場合のエラー情報や処理終了を意味する DONE（ダン）ビットを含んでいます．MSDOS．SYS は上述した“I/O リクエストコール”を IO．SYS へ発することによって各デバイスと情報を交換し，

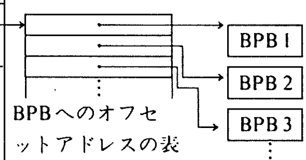
注） ファンクションコールの 44H“I/O コントロール”は I/O リクエストコールのコマンド番号 3 と 12 を利用したもので，具体的利用の仕方は，“MSDOS．SYS 解析”の項を参照のこと．

	オフセット	内容		
ES:BX→	+0	レコード長(コマンドパケット全体のバイト数)	リクエストヘッダ	コマンドパケット
	+1	論理装置コード	リクエストヘッダ	コマンドパケット
	+2	コマンド番号(図2・7)	リクエストヘッダ	コマンドパケット
	+3,4	ステータス	リクエストヘッダ	コマンドパケット
	+5~+C	システムリザーブ(領域だけ確保していればよい)	リクエストヘッダ	コマンドパケット
		各コマンドによって0~9バイト可変 内容も各コマンドによって異なる	引数エリア	コマンドパケット

引数エリア(コマンド番号別)
D…デバイスドライバ側で値をセットする
M…MS-DOS.SYSがI/Oリクエストを発行するときセットする

コマンド番号0

+D	デバイスドライバがサポートしているBPB数	D
+E~+11	デバイスドライバのRAM上の最終アドレス	D
+12~+15	BPBへのポインタの表のアドレス → デバイスドライバにイニシャライズのためのエントリ時はCONFIG.SYSファイルのDEVICE=のあとのアドレスがここに書かれる。 (くわしくは第1章CONFIG.SYS解析のDEVICE処理参照)	D M



コマンド番号1

+D	メディアディスクリプタ※1	M
+E,F	−1……ディスク交換されている 0……交換されたか不明 1……交換されていない	D

※1 メディアディスクリプタ
デバイスドライバ内でBPBがいくつかあるときの識別番号。
(ふつうドライバ内の何番目のBPBかというふうに番号をつけるとよい。)

コマンド番号2

+D	メディアディスクリプタ※1	D
+E~+11	ディスク判別用のセクタを読み込んだバッファへのアドレス	M
+12~+15	BPBのアドレス	

アドレス—MS-DOS標準(デバイスヘッダの属性bit13=0)ならFATの第1セクタのこと。もし標準以外ならバッファへは読み込まれていない。

コマンド番号3,4 8,9,12

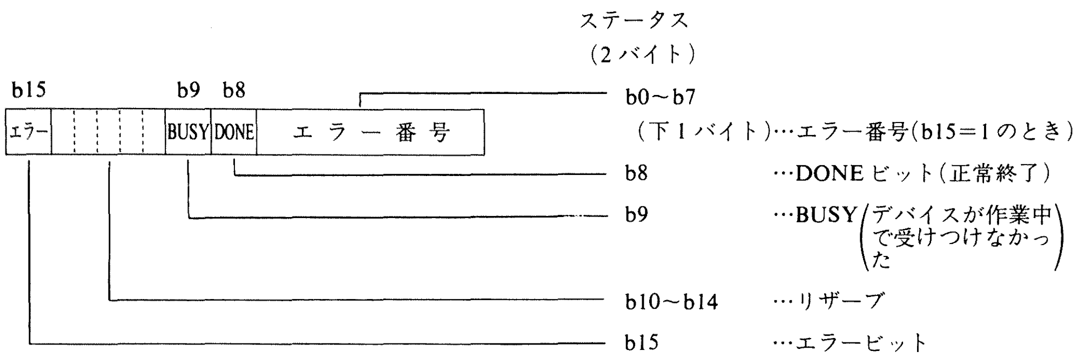
+D	メディアディスクリプタ※1	M
+E~+11	入出力バッファアドレス	M
+12,13	コマンド番号 3,12…データバイト数 4,8,9…セクタ数	M/D
+14,15	開始セクタ番号(ブロック型デバイスのみ)	M

コマンド番号5

+D	読み込んだデータ	D

コマンド番号6,7,10,11は引数エリアはない

図2.8 コマンドパケット



エラー番号	内容	対応するCOMMAND.COMエラーメッセージ
0	ディスクがライトプロテクトされている	書込み禁止です
1	論理装置コードに対するデバイスが存在しない	ドライブ指定が違います
2	ドライブの準備ができていない	ドライブの準備ができていません
3	コマンド番号が間違っている	無効なコマンドです
4	CRC(Cyclic Redundancy Code)エラー	データエラーです
5	コマンドパケットの長さが間違っている	要求形式が違います
6	シークエラー	シークエラー
7	メディアタイプが間違っている	このディスクは扱えません
8	セクタが存在しない	セクタが見つかりません
9	プリンタ用紙切れ	用紙がありません
10	ライトエラー	書き込みができません
11	リードエラー	読み取りができません
12	その他のエラー	エラーです

図 2.9 ステータスの内容とエラー番号

データ入出力以外にもディスクの交換のチェックやデバイスのステータスのチェックなどを行うことができます.

ステータスのビット0～7はエラービット15が1のときのみ意味があります. I/Oリクエストコール終了時のビット8と15の状態は

	bit15	bit8
通常終了 (エラーなし)	0	1
エラー終了時	1	1

とエラー終了にかかわらず, DONEビット (bit8) は1になります.

なお, I/Oリクエストコールのコマンド番号は0が初期化, 1～6までが入力関係, 7～12までが出力関係となっています.

以下でIO.SYSに標準で組み込まれているデバイスについて説明しますので, CONFIG.SYSによるユーザーデバイスの追加などの参考にしてください.

2・4 I/O リクエストコールの実際

IO.SYS の中には CON, AUX, PRN, CLOCK, ディスクの5つのデバイスドライバが標準として組み込まれていますが，これらのドライバは前節で述べた I/O リクエストコールをどのように処理しているのでしょうか？　自分で CONFIG.SYS でシステムに登録するデバイスドライバを製作するときにも参考になると思います．

まず，**図2･7**に I/O リクエストコールのコマンドを掲げましたが，IO.SYS では全コマンドをサポートしているわけではありません．必要ないものはステータスの DONE ビットを1にするだけでリターンしたりします．たとえば，CON, AUX, PRN, CLOCK のようなキャラクタ型デバイスに対して BPB は存在しませんので，“BPB 作成（コマンド2番）”では DONE ビットを1にするだけでリターンします．**図2・10**に実際の処理をするものに○，DONE ビットだけを立てて（1にして）リターンするものに×をつけて示します．表の示すように”I/O コントロール機能”はサポートされていません．したがってファンクションコール 44H の“デバイスに対する I/O コントロール”を実行しても，デバイスヘッダの属性ビット I/O コントロール・サポートビット b14（**図2･3**参照）をチェックされエラーリターンします．

I/O リクエストコールを処理するためにはデバイス(ハードウェア)との情報のやりとりが必要ですが，このためには一般に I/O ポートをアクセスしなければなりません(**図1･2**参照)．しかし PC-9801 には ROM-BIOS (ROM-BASIC INPUT OUTPUT SYSTEM) と呼ばれるハードウェアに対するハンドラが ROM の中に書き込まれているので（**図1・1**参照），IO.SYS のデバイスドライバは，ハードウェアとのやりとりをほとんどこの ROM-BIOS を使って処理をしています．

ROM-BIOS は INT コールを使って呼び出します．各デバイスと対応する INT コールはそれぞれつぎのようになっています．

コマンド ＼ デバイス	0	1	2	3	4	5	6	7	8	9	10	11	12
	初期化	メディアチェック	BPB 作成	IOCTL 入力	リードデータ	非破壊型入力	入力用ステータスチェック	入力用バッファフラッシュ	ライトデータ	ライト&ベリファイ	出力用ステータスチェック	出力用バッファフラッシュ	IOCTL 出力
CON	×	×	×	×△	○	○	×	○	○	○[注2]	×	×	×
AUX	×	×	×	×△	○	○	×	○	○	○[注2]	○	×	×
PRN	×	×	×	×△	×	×[注1]	×	×	○	○[注2]	○	×	×
CLOCK	×	×	×	×△	○	×[注1]	×	×	○	○[注2]	×	×	×
ディスク	○	○	○	×△	○	×[注1]	×	×	○	○	×	×	×

○ …サポートしている
× …DONE ビットを1にするのみ
△ …コマンド番号が間違っている（エラー番号3）のエラー出力と DONE ビットを1にするのみ
注1…BUSY ビットと DONE ビットを1にするのみ
注2…ライトデータと同じ処理（処理アドレスが同じ）

図2・10　IO.SYS サポートデバイスの I/O リクエストコールのサポート状況

CON ………INT	18H	キーボード，CRT BIOS
AUX ………INT	19H	RS-232C BIOS
PRN ………INT	1AH	カセット注)，プリンタ BIOS
CLOCK ……INT	1CH	カレンダ・インターバルタイマ BIOS
ディスク……INT	1BH	フロッピーディスク BIOS

2・5　各デバイス共通ルーチン

各デバイスのストラテジエントリ注)は共通です．ES：BX にコマンドパケットの先頭アドレスが入っていますから，ES と BX の値を IO.SYS のデバイスドライバのワークエリアに保存します．すなわち一般のデバイスドライバとおなじように，以下のような処理をしています．

```
Strategyentry proc Far
            MOV   COMPAK_OFS, BX
            MOV   COMPAK_SEG, ES
            RET
Strategyentry endp
```

つぎに割り込みエントリ注)は各ルーチンに分かれていますが，**図 2・11** のフローチャートのようにコマンドアドレステーブル方式で各ルーチンへ処理が移されるようになっています．

2・6　デバイス CON

CON で実際に処理を行っているのは I/O リクエストコマンド 4，5，7，8，(9) です（**図 2・10** 参照）．それぞれリードデータ(4)，非破壊型入力(5)，入力用バッファフラッシュ(7)，ライトデータ(8)，（ライト & ベリファイ(9)）です．以下説明していきますが，コマンドパケットセグメント先頭アドレスを COMPAK とします．

2・6・1　CON リードデータ（コマンド番号 4）

まず読み込むバイト数が COMPAK＋12H，転送アドレスが COMPAK＋0EH に入っているので

```
MOV  CX,   COMPAK+12H
LES  DI,   COMPAK+0EH
```

でレジスタにセットします．LES DI，という命令はメモリに

下位アドレス	上位アドレス
オフセット	セグメント

注) ストラテジエントリと割り込みエントリについてはデバイスヘッダの項（35 ページ）参照．

と入っていることを仮定し，オフセットを DI，セグメントを ES にロードする命令です．LES は "Load ES" ということです．

つぎに

```
CON_READ : PUSH  CX
           PUSH  DI
```

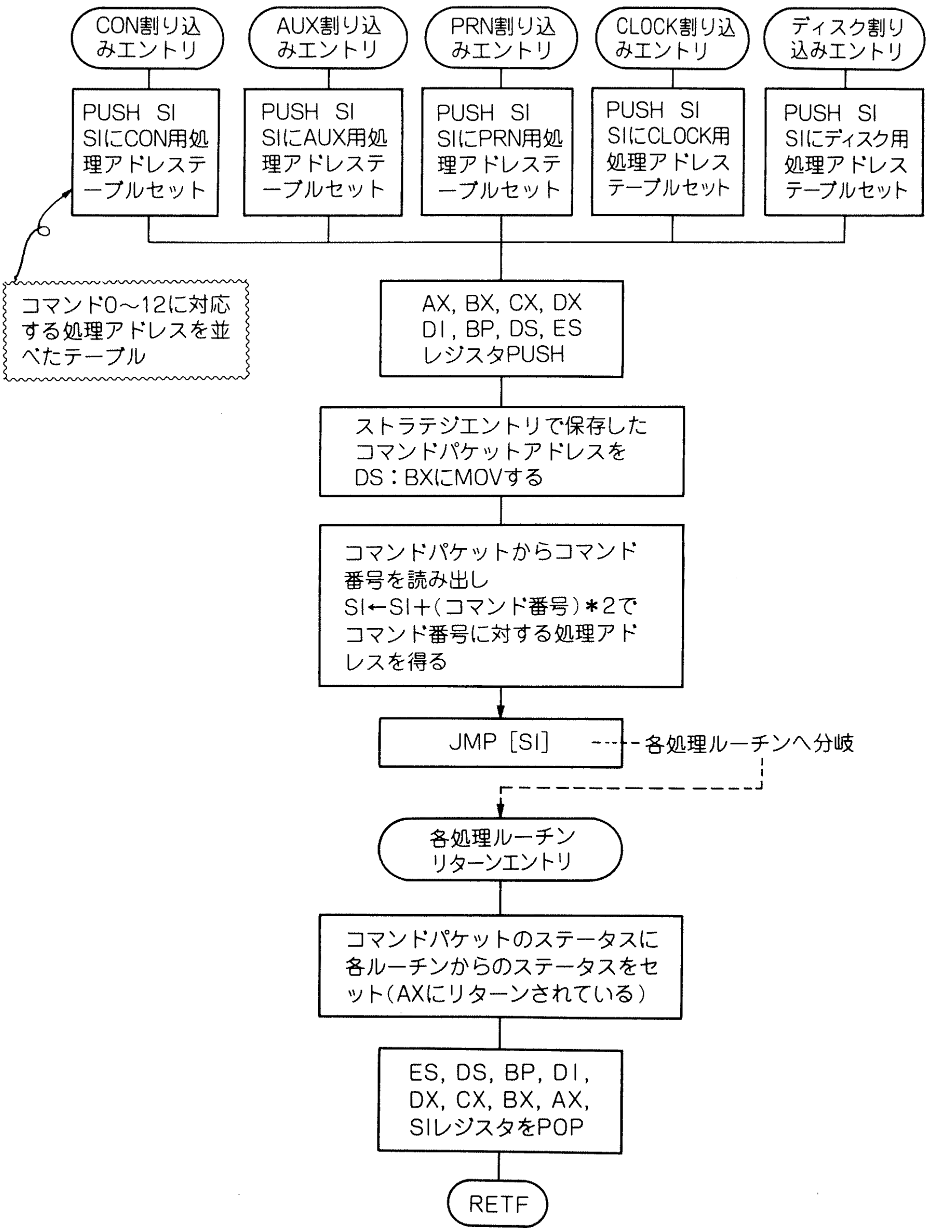


図 2・11 IO.SYS 組み込みデバイスドライバ割り込みエントリ分岐部分

注) カセットは現在サポートされていない．

```
CALL   CON 1バイト入力ルーチン
POP    DI
POP    CX
STOSB
LOOP   CON_READ
MOV    AH, 1   ;   Done ビット=1
JMP    各処理ルーチンリターンエントリ（図2・11）
```

CON1バイト入力ルーチンを読み込むバイト分コールし，STOSBでコール元の転送バッファに書き込みます．

先行入力

先行入力というのは，MS-DOSがコマンド実行中であっても割り込みを利用してキーボード入力を16文字分バッファリングできる機能のことです．リングバッファはセグメント0000H，オフセット502H～521Hの32バイト（1文字に2バイトを使っている）を

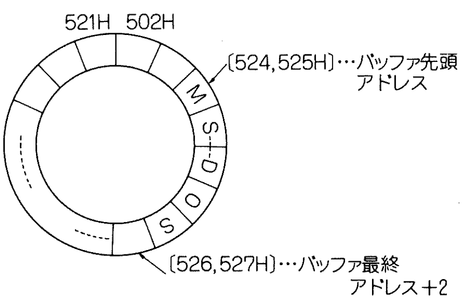


図 A

図Aのように環状（リング）につなげて，バッファリングで521Hまで使用したらふたたび502Hから始まるというようにバッファを使います．バッファの先頭アドレスは[524, 525H]，最終アドレス+2は[526, 527H]，バッファリング文字数は[528H]に格納されていて，図Bのように[526, 527H]が[524, 525H] を追い越すことはできません．

先行入力は，ROM-BIOSが管理しています．

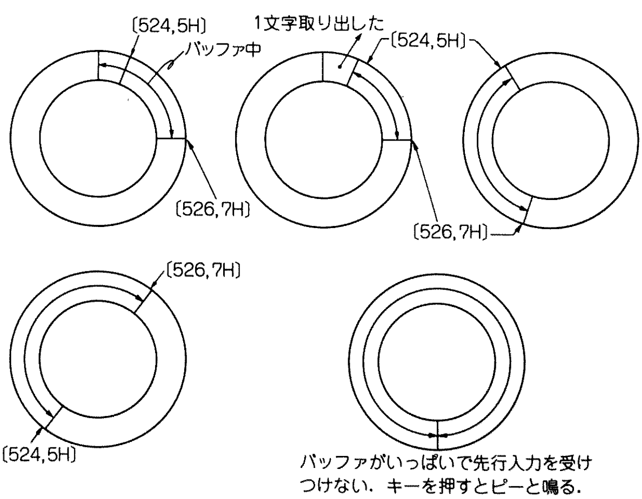


図 B

CON1 バイト入力ルーチンでは“CON 非破壊入力”を CALL して，先行入力がある場合はキー入力リングバッファから 1 文字取り出し，先行入力がなければ INT 18H でキー入力します．

入力したコードが CTRL-P[注1] または CTRL-N（それぞれ JIS コード 10H，0EH）のときは，画面出力と同時にプリンタへ出力するフラグをトグル[注2]で ON／OFF します．入力した文字は AL

INT 18H キーボード，CRT ROM-BIOS について

ROM-BIOS は AH レジスタにコマンド番号を入れて呼ぶようになっていますが，INT 18H はキーボードに対してコマンド 0～4，テキスト画面に対して 0AH～1AH，グラフィック画面に対して 40H～4AH の値が割り当てられています．IO.SYS で使用しているのはこのうちコマンド 0 と 2，初期化のときに 1AH です．それぞれ

0 ………キーコードリード
2 ………キーセンス
1AH……ユーザー漢字フォントパターンセット

の機能を持っています．

また上記コマンドがすべて使えるかというと，カーソル表示に関しては IO.SYS 内で管理しているのでカーソルに関するコマンド 0FH～13H は使えません．グラフィック画面に関するコマンドも初期状態では使えませんが，ワークエリアの初期設定をしてやれば使えるようになります．ただ単にグラフィック V-RAM を画面に表示させ CPU で直接グラフィック V-RAM をアクセスして描画を行うだけなら，コマンド 40H でグラフィック画面表示状態となります．41H でグラフィック画面表示停止とすることもできます．

さて，IO.SYS で使用しているコマンド 0，2，1AH についてくわしく説明しておきます．

① コマンド 0（キーコードリード）

このコマンドを実際実行するには，マシン語で

```
MOV AH, 0 ; コマンド番号
ITN 18H
```

とします．キーボードの入力があるまで待ちますが，入力があれば AH に押されたキーに対応する“内部コード”が，AL に“JIS キーコード”が入れられてリターンします．AH の内部キーコードはファンクションキーや XFER キーなどの特殊キーを判別するために使用しています．また，CTRL キーと XFER キーなど特殊キー同士を同時に押したことの判別も内部コードに反映されるようになっています．キーボードのキーと内部コードの対応表を図 **2・12** に示します．

② コマンド 2（キーセンス）

コマンド 0 と同じように

```
MOV AH, 2 ; コマンド番号
ITN 18H
```

とすると，AL の対応する

b0……SHIFT キーが押されているとき 1
b1……CAPS キーが押されているとき 1
b2……カナキーが押されているとき 1
b3……GRAPH キーが押されているとき 1
b4……CTRL キーが押されているとき 1

のビットが 1 となります．IO.SYS ではカナキーが押されているかどうかの判別に使用しています．

上位4ビット → 列 / 下位4ビット → 行

下＼上	0	1	2	3	4	5	6	7	8 (SHIFT +)	9 (CTRL +)	A (SHIFT +)	B (CTRL +)
0	ESC	Q タ	F ハ	, < ネ ,	—	.						
1	1 ! ヌ	W テ	G キ	. > ル 。	／							
2	2 " フ	E イ ィ	H ク	／ ? メ ・	7		f・1		f・1	f・1		
3	3 # ア ァ	R ス	J マ	— ロ	8		f・2		f・2	f・2		
4	4 $ ウ ゥ	T カ	K ノ	スペース スバー	9		f・3		f・3	f・3		
5	5 % エ ェ	Y ン	L リ	XFER	*		f・4		f・4	f・4	XFER	XFER
6	6 & オ ォ	U ナ	; + レ	ROLL UP	4		f・5		f・5	f・5		
7	7 ' ヤ ャ	I ニ	: * ケ	ROLL DOWN	5		f・6		f・6	f・6		
8	8 (ユ ュ	O ラ	] ム 」	INS	6		f・7		f・7	f・7		
9	9) ヨ ョ	P セ	Z ツ ッ	DEL	+		f・8		f・8	f・8		
A	0 ワ ヲ	@ ゛	X サ	↑	1		f・9		f・9	f・9		
B	− ＝ ホ	[゜ 「	C ソ	←	2		f・10		f・10	f・10		
C	^ ヘ	↲	V ヒ	→	3							
D	¥ _	A チ	B コ	↓	=							
E	BS	S ト	N ミ	HOME CLR	0						HOME CLR	
F	TAB	D シ	M モ	HELP	,							

（例）。 [P セ] を押すと上位4ビットが1, 下位4ビットが9なので AH に 19H が入る.

。 [CTRL]＋[XFER] と押すと AH には B5H が入る.

（注）3EH～50H はテンキー

図 2.12　INT 18H キーコード入力の内部コード表

③　コマンド 1AH（ユーザー漢字フォントパターン・セット）

```
MOV BX, バッファセグメント
MOV CX, バッファオフセット
MOV DX, 漢字 JIS コード
MOV  AH, 1AH ; コマンド番号
INT   18H
```

とすると BX：CX で指定されたフォントパターン・バッファ（32バイト）が，DX で指定された JIS 漢字コードの漢字バッファに登録されます．ただし，ユーザーフォントパターン定義用として予約されている JIS漢字コードは，7621H～765FH の領域です．このコマンドはシステム初期化のときに USKCG. SYS という漢字フォントファイルから定義フォントパターンを読み込むときに使用されます．USKCG. SYS は USKCG. COM というユーザー定義フォント管理外部コマンドによって作成することができます．

レジスタに入れてリターンします.

2・6・2 CON 非破壊入力（コマンド番号 5）

非破壊入力というのは，先行入力によってキー入力リングバッファに蓄えられたキーコードを，バッファの先頭アドレスポインタやバッファリング文字数カウンタを更新することなく取り出すことです.

入力文字がないときには BUSY フラグを 1 にして終了します. 入力文字があるときは取り出した文字コードをコマンドパケットの +0DH に格納して BUSY フラグを 0 にして終了します. すなわちコマンドパケットの先頭アドレスを COMPAK とすると，入力ありのとき

```
MOV COMPAK+0DH, AL   ;        文字コード
MOV COMPAK+03H, 01H  ;        NonBusy, Done
```

となります.

CON の非破壊入力は単に文字コードを返すだけでなく特殊キー（ファンクションキーやコントロールキーなど）が押されたときの処理や，日本語入力時のサポートも行っています.

入力モードは大きく分けて

・通常入力モード（JIS8 ビットコード）

・日本語入力モード

の 2 つがあります. 日本語入力モードには CTRL+XFER 同時押下で行き来できます. 各モードの下では入力キーコードによってさらにいくつもの処理アドレスに分岐するようになっていますが，ここは言葉でだらだら説明するよりも，**図 2・13** のフローチャートを見てもらったほうがわかりやすいでしょう.

CON 非破壊入力ルーチンの出力条件はゼロフラグ（ZF）=1 のとき入力なし，0 のとき入力ありで，入力文字コードは AL に入れられてリターンします. このリターン値を使って，さきに述べたように，コマンドパケットに値をセットします.

2・6・3 CON 入力バッファフラッシュ（コマンド番号 7）

ROM-BIOS のキー入力バッファをクリアします. ROM-BIOS のキー入力バッファはセグメント 0000H のオフセット 502H～521H にあり，そのポインタは

524, 525H……キーバッファ先頭ポインタ

526, 527H……キーバッファエンドポインタ

528H …………バッファリング中の文字数

となっています.（524, 525H),（526, 527H）に 502H を，（528H）に 00H をセットしてバッファクリアとしますが，ROM-BIOS はつねに割り込みを利用して先行入力をしていますので，いったん割り込み禁止コマンド CLI を実行してから上記クリアを行います. **図 2・13** を見てもわかるように，ROM-BIOS のキーバッファに加えて CON デバイスドライバ自体も入力バッファを持っていま

注1) アルファベットキーと CTRL キーを同時に押すと，アルファベットのコード−40H の値となります. “P”は 50H, “N”は 4EH なのでそれぞれ 50H−40H=10H, 4EH−40H=0EH となります.

注2) キーが押されたときフラグが ON の場合は OFF にし，OFF のときは ON にするような動作を “トグル” といいます. 実際のプログラムでは XOR コマンドを用いて XOR FLAG, 1 としています.

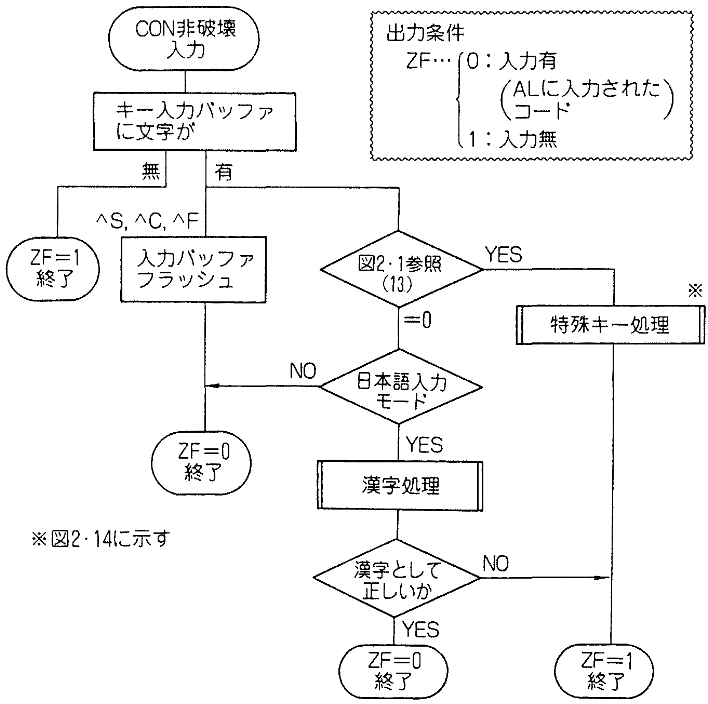


図 2・13　CON 非破壊入力フローチャート

すので，ROM-BIOS のキーバッファと同時に CON 入力バッファもクリアしています．

クリアが終了したらふたたび割り込み許可コマンド STI を実行し，コマンドパケットのステータスの DONE ビットを 1 にしてリターンします．

2・6・4　CON ライトデータ（コマンド番号 8）

コンソールの CRT に文字（1/2 角〜漢字）を出力します．文字を単に出力するだけでなくカーソルの制御や文字の色づけ，ファンクションキーの処理もすることができます．

コマンドパケットの先頭アドレスを COMPAK とすると

```
          LES    SI, COMPAK+0EH
          MOV    CX, COMPAK+12H
CONWRT :  ES :   LODSB
          MOV    CL, AL
          CALL   1バイト出力ルーチン
          LOOP   CONWRT
          MOV    AH, 01H
          JMP    終了ルーチンへ
```

という処理をします．ここで“1 バイト出力ルーチン”というのは，CRT への 1 バイト出力処理を行うルーチンのことです．漢字なら，1 バイトをバッファリングしてつぎに 1 バイト出力ルーチン

内部コード (Hex)	機能	
00〜2F	RET	
30〜34	RET	
35	[XFER]キー 日本語入力モードフラグ ON なら変換処理へ	
36, 37	[ROLL UP] [ROLL DOWN] 日本語入力モードでなければキー内容表示	
	日本語間接入力モード	通常入力モード
38	[INS]行編集インサートフラグ ON	キーに登録された内容表示
39	[DEL]行編集 1 文字削除	
3A	[↑]前漢字候補	
3B	[←]漢字候補カーソル左移動	
3C	[→]　〃　右 〃	
3D	[↓]次漢字候補	
3E	[HOME CLR]編集行の先頭へカーソルを移動する	
3F	[HELP]　RET	
40〜5F	RET	
60, 61	RET	
	日本語入力モード	通常入力モード
62	[f·1] 間接入力モード切換	ファンクションキーの内容表示
63	[f·2]〔英数〕R〔かな〕（ローマ字かな変換）切換	
64	[f·3]〔かな〕〔カナ〕切換	
65	[f·4] ＊半角，全角切換	
66	[f·5]〔16 進〕コード入力モード	
67〜6F	[f·6]〜[f·10]　RET	
70〜7F	RET	
80〜8F	[SHIFT]＋[f·n]裏ファンクションキーの内容表示	
90〜94	[CTRL]＋[f·1]〜[f·3]　RET	
95	[CTRL]＋[f·4]グラフィック文字を受けつけるようにするが日本語入出力不可にする（トグル） すなわち，XOR 図 2・1 の（13），1	
96	[CTRL]＋[f·5] 1 バイトコードを 16 進入力	
97	[CTRL]＋[f·6] 20 ↔ 25 行モード切換（トグル）	
98	[CTRL]＋[f·7]ファンクションキーガイドライン表示（表・裏・未表示）選択	
99	[CTRL]＋[f·8]テキスト画面クリア	

9A	CTRL+f·9 スクロールの速度（速・遅）切換（トグル）
9B～9F	CTRL+f·10　RET
A0～AF	AE SHIFT+HOME CLR 日本語入力モードの間接入力行クリア
B0～BF	B5 CTRL+XFER 日本語入力↔通常入力モード切換（トグル）図 2・1の（13）=0のときは無効
C0～FF	RET

（注）処理内容 RET と書いているのはなにもせずただリターンするもの．

図 2・14　キー内部コード処理ルーチン対応表

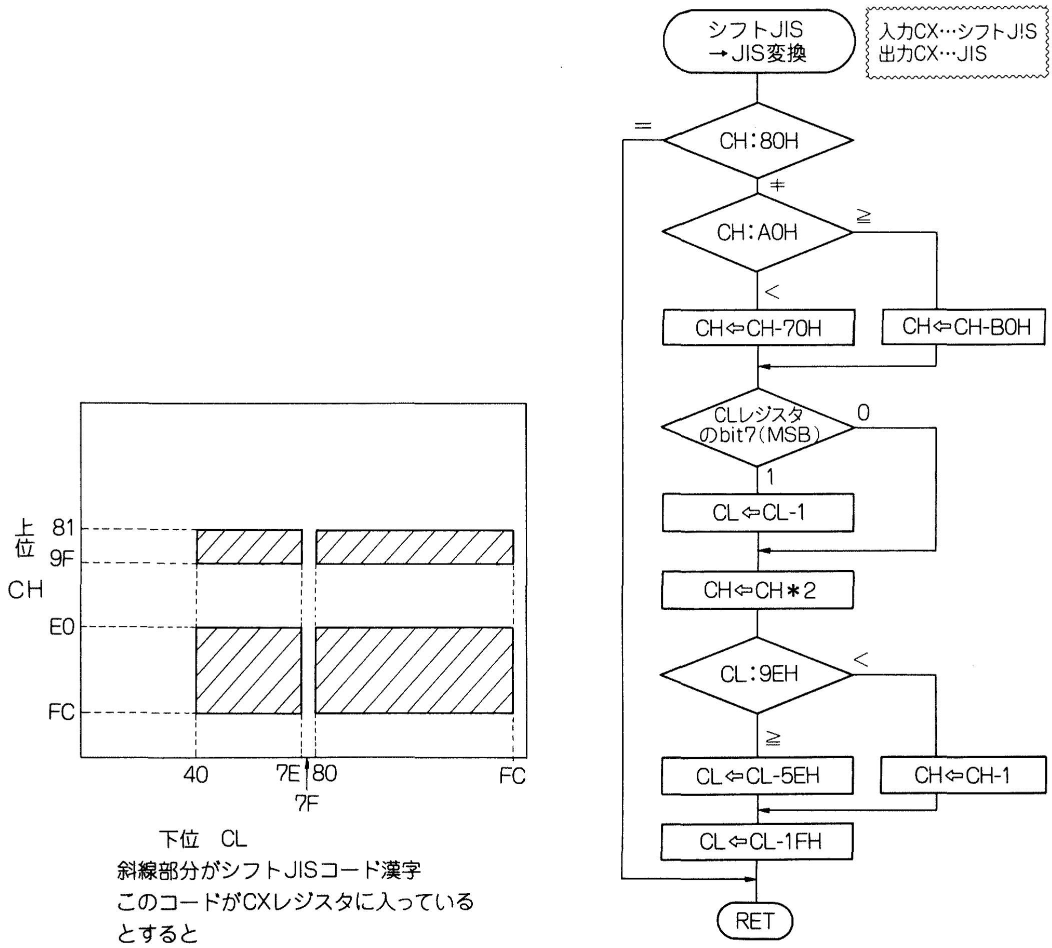
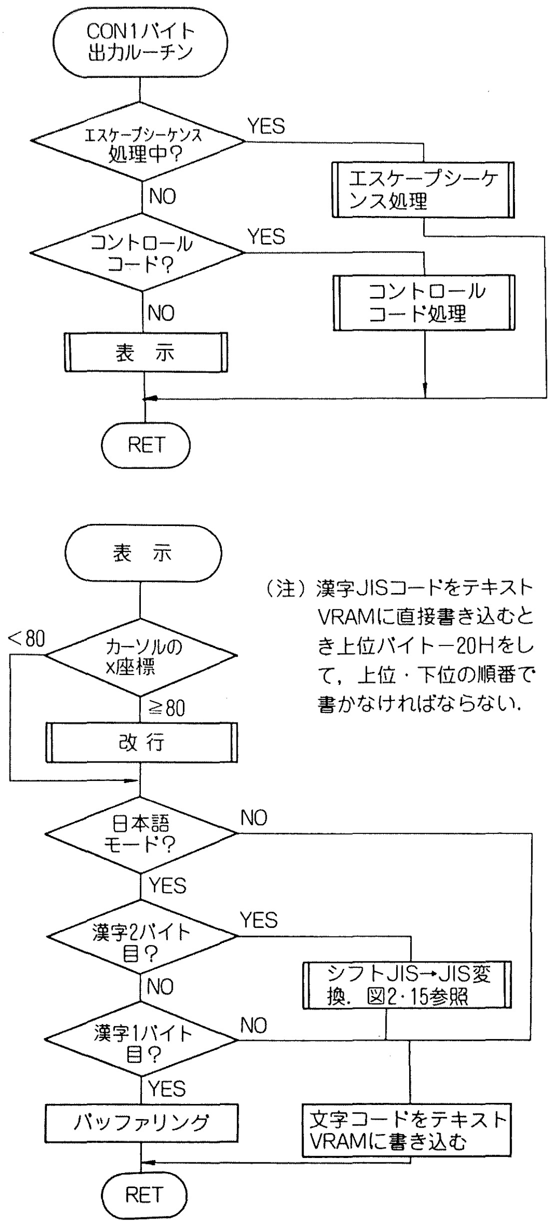


図 2・15　シフト JIS → JIS 漢字コード変換

に渡された1バイトと組み合わせて漢字1文字を出力します．漢字コードの1バイト目はマイクロソフトのシフト JIS コード体系になっていますので，81H 以上 A0H 未満，または E0H 以上 FDH 未満となっていることで判断します．この領域は1/2角8ビット符号系で，PC-9801 はグラフィック文字領域となっていますので，漢字とグラフィック文字は同時には使えないようになっているのです（**図 2・1** の（13），**図 2・14** CTRL + f·4 参照）．

CRT への出力は JIS 漢字コードしか受け付けませんので，入力されたシフト JIS 漢字コードは**図 2・15** のようなアルゴリズムで JIS 漢字コードに変換され，CRT へ出力されます．もちろん入力さ



（注）漢字JISコードをテキストVRAMに直接書き込むとき上位バイト−20Hをして，上位・下位の順番で書かなければならない.

コントロールコード処理テーブル

コード	対応キー	処理内容
7	CTRL+G	BEEP
8	CTRL+H BS ←	カーソルより前の1文字削除
9	CTRL+I TAB	8タブ処理
A	CTRL+J ↓	ラインフィード
B	CTRL+K ↑	カーソルを1行上にあげる
C	CTRL+L →	カーソルを1文字右へ進める
D	CTRL+M ↵	復帰改行
1A	CTRL+Z	画面クリア
1B	CTRL+[ESC	エスケープシーケンス初期化処理*
1E	CTRL+^	画面のホーム位置にカーソル移動
その他	コードそのままをテキストVRAMに表示する	

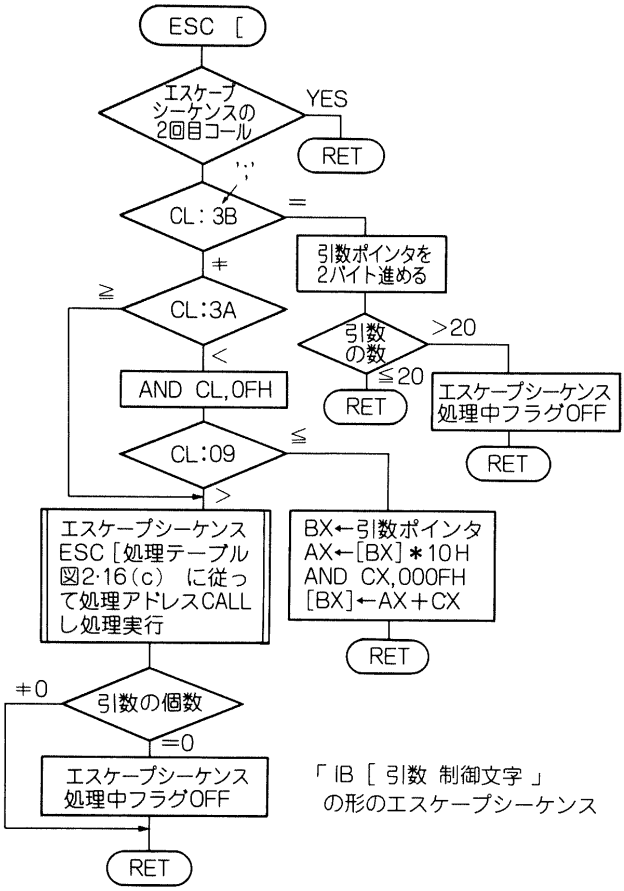
*エスケープシーケンス初期化処理とはエスケープシーケンス用引数バッファエリアをクリアしエスケープシーケンス処理中フラグをONにすること.

図 2・16 (a) CON1バイト出力ルーチン

れた1バイトが漢字コードでなかったらつぎの2つの場合に分かれます. すなわち20Hより大きいか小さいかです. 20Hより大きければその1/2角文字をそのまま出力しますし，20H未満ならばコントロールコードの処理をします. コントロールコードの中で1BH（ESC）は通称エスケープコードといって，カーソルの制御やファンクションキー表示停止などのできるエスケープシーケンス機能を持っています. これらのコードの処理の様子を**図2・16（a）～（c）**にまとめました.

図2・16（c）については実例をあげて説明します. カーソルの位置指定ESC［py；pxHでたとえばx座標23文字目，y座標7行目の位置を指定したいときは，py='7'，px='23'となるので

ESC	[	7	;	2	3	H
1BH	5BH	37H	3BH	32H	33H	48H



エスケープシーケンス処理テーブル

コード	文字	処 理 内 容
5B	[	図 2・16 (b) 参照
3D	=	引数 l, c をとって LOCATE c, l
2A	＊	画面クリア ESC [2J と等価
28	(	
44	D	ラインフィード
45	E	復帰改行
4D	M	カーソルを 1 行上に移動
29	)	引数 n をとる n=0 のとき g モード解除(注) 3 のとき g モード設定
その他		エスケープシーケンス 処理中フラグ OFF

g モードとはグラフィック文字を有効にするが漢字は表示できなくなるモードである.

図 2・16 (b)　エスケープ [(ESC [) 処理

というコードを送ります. 上段に書いたのが文字記号で下段がそれに対する 16 進コードです. このようにアスキー表現の 10 進数字列と解釈してください. MS-DOS のファンクションコールの 09H がコンソールに文字を出力する場合 CON 出力ルーチンを呼び出しますので, ファンクション 09H を使ってこの文字列を出力する例を**図 2・17** に示します.

なお, pn, py, px, ps などの記号は ASCII 表現の文字列で, たとえば 10 進で 23 を表わしたいときは '23' とします. すなわち, ASCII コードで 32H, 33H となります.

たとえば**図 2・17** では y 座標を 23 としているので, ダンプリストと合せて見てください.

```
A>DEBUG
-A
10EF:0100 MOV DX,200
10EF:0103 MOV AH,9
10EF:0105 INT 21
10EF:0107
-E200 1B 5B 37 3B 32 33 48 41 24
-D200,20F
10EF:0200  1B 5B 37 3B 32 33 48 41-24 00 00 00 00 00 00 00   .[7;23HA$.......
-G=100,107
7;23HA
AX=0924  BX=0000  CX=0000  DX=0200  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=10EF  ES=10EF  SS=10EF  CS=10EF  IP=0107   NV UP DI PL NZ NA PO NC
10EF:0107 45            INC     BP
-Q

A>
```

図 2・17　ESC [7 ; 23H　実行例 (LOCATE23, 7 をして文字 A を出力している)

<table>
<tr><th colspan="2">コード</th><th rowspan="2">引数</th><th colspan="2" rowspan="2">処 理 内 容</th></tr>
<tr><th>16進コード</th><th>文字</th></tr>
<tr><td>41</td><td>A</td><td>pn</td><td colspan="2">カーソルを pn 行上に移動</td></tr>
<tr><td>42</td><td>B</td><td>pn</td><td colspan="2">カーソルを pn 行下に移動</td></tr>
<tr><td>43</td><td>C</td><td>pn</td><td colspan="2">カーソルを pn 文字右に移動</td></tr>
<tr><td>44</td><td>D</td><td>pn</td><td colspan="2">カーソルを pn 文字左に移動</td></tr>
<tr><td>48</td><td>H</td><td>py ; px</td><td colspan="2">カーソル位置指定 LOCATE px, py (x, y は座標が逆になる)</td></tr>
<tr><td>4A</td><td>J</td><td>pn</td><td colspan="2">pn=0…カーソル～行末クリア
=1…行頭～カーソル位置までクリア
=2…画面クリア</td></tr>
<tr><td>4B</td><td>K</td><td>pn</td><td colspan="2">pn=0, 1…ESC [pnJ とおなじ
=2 …カーソルの位置する行をクリア</td></tr>
<tr><td>4C</td><td>L</td><td>pn</td><td colspan="2">pn 行空白行を挿入 (カーソル行の下)</td></tr>
<tr><td>4D</td><td>M</td><td>pn</td><td colspan="2">pn 行削除 (カーソル行より下)</td></tr>
<tr><td>6D</td><td>m</td><td>ps ; ～ ; ps</td><td colspan="2">文字属性 (色, ブリンク etc) 指定</td></tr>
<tr><td>66</td><td>f</td><td>py ; px</td><td colspan="2">LOCATE px, py ESC [py ; pxH とおなじ</td></tr>
<tr><td rowspan="2">68</td><td rowspan="2">h</td><td>>pn</td><td colspan="2">pn=1…ファンクションキー表示許可
3…20 行モードセット
5…カーソル消去</td></tr>
<tr><td>? 7</td><td colspan="2">1 行の文字数 (80 文字) を越えた表示は改行して表示する</td></tr>
<tr><td rowspan="2">6C</td><td rowspan="2">l</td><td>>pn</td><td colspan="2">pn=1…ファンクションキー表示禁止
3…25 行モードセット
5…カーソル表示</td></tr>
<tr><td>? 7</td><td colspan="2">1 行の文字数 (80 文字) を越えた表示は無視する. 改行しない</td></tr>
<tr><td>6E</td><td>n</td><td>6</td><td colspan="2">カーソル位置をコンソールに知らせる(注)</td></tr>
<tr><td>73</td><td>s</td><td>な し</td><td colspan="2">カーソル位置と属性を一時退避する</td></tr>
<tr><td>75</td><td>u</td><td>な し</td><td colspan="2">一時退避したカーソル位置と属性を復帰する</td></tr>
<tr><td>3E</td><td>></td><td>な し</td><td>>フラグ ON</td><td rowspan="3">ESC [>や ESC [? という文字列をシーケンスに使っている処理へのフラグ</td></tr>
<tr><td>3F</td><td>?</td><td>な し</td><td>? フラグ ON</td></tr>
<tr><td>3D</td><td>=</td><td>な し</td><td>=フラグ ON</td></tr>
<tr><td colspan="2">その他</td><td>な し</td><td colspan="2">なにもせず, すぐ RET</td></tr>
</table>

・pn, py, px, ps はアスキー文字列の 10 進数

(注) ESC [n を実行したあと CON リードを 8 バイト分行えば

1B 5B yy 3B xx 52

ESC [py ; px R

の形式でカーソルの x, y 座標が読み出される (ここで xx…X 座標, yy…Y 座標, xx, yy は 2 桁の 10 進数アスキー表現)

図 2・16 (c) エスケープシーケンス ESC [処理テーブル

```
A>DEBUG
-A
10EF:0100 MOV DX,200
10EF:0103 MOV AH,9
10EF:0105 INT 21
10EF:0107 MOV CX,8 ……8バイト分入力（カーソル座標情報）
10EF:010A MOV DI,210 ……カーソル座標情報をたくわえるためのバッファ
10EF:010D MOV AH,7 ……1バイト CONSOLE 入力
10EF:010F INT 21
10EF:0111 STOSB
10EF:0112 LOOP 10D
10EF:0114
-F200,21F,0
-E200 1B 5B 36 6E 24
-D200,21F
10EF:0200  1B 5B 36 6E 24 00 00 00-00 00 00 00 00 00 00 00   .[6n$...........
10EF:0210  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00   ................
-G=100,114
6n
-R
AX=0752  BX=0000  CX=0000  DX=0200  SP=FFEE  BP=0000  SI=0000  DI=0218
DS=10EF  ES=10EF  SS=10EF  CS=10EF  IP=0114   NV UP DI PL NZ NA PO NC
1580:0100 03E8          ADD     BP,AX
-D200,21F
10EF:0200  1B 5B 36 6E 24 00 00 00-00 00 00 00 00 00 00 00   .[6n$...........
10EF:0210  1B 5B 32 34 3B 30 31 52-00 00 00 00 00 00 00 00   .[24;01R........
-Q                 y座標24
A>        |<-------------------------->|
                     8バイト
```

図 2・18 ESC [6n 実行例（コマンド発行後コンソール入力を 8 バイト分行うことによって ESC [y 座標；x 座標 R の形でカーソル座標を得ることができる）

もう 1 つ，ESC［6n（カーソル位置をコンソールに知らせる）の実行例を**図 2・18** に示します．カーソルの x，y 座標を CON 入力のバッファにセットしてカーソル座標情報を知らせますので，このコマンド発行後すぐにキー入力ファンクションコールを発行して 8 バイト分の文字列を入力しなければなりません．もしキー入力をせずにユーザーキー入力ルーチンに入ると，カーソルの座標情報は CRT（標準出力）へ出力されてしまうことになります．

コマンド発行後コンソール入力を 8 バイト分行うことによって，ESC［y 座標 ；x 座標 R の形でカーソル座標を得ることができます．

「ESC［ps；ps；…；ps m」の文字属性の指定の ps の値と，色やブリンク，リバースの対応については**図 2・19** にまとめました．

このエスケープシーケンスを使うことによって，テキストエディタやメニュー画面などを容易に作成することができるでしょう．

2・7 デバイス AUX

AUX で実際に処理を行っているのは 4，5，7，8，(9)，10 です（**図 2・10** 参照）．それぞれリードデータ(4)，非破壊入力(5)，入力用バッファフラッシュ(7)，ライトデータ(8)，(ライトデータ & ベリファイ(9))，出力用ステータスチェック(10) です．処理内容はほとんど CON と同じですが，ハードウェアとデータをやりとりするのは INT 19H の RS-232C ROM-BIOS です．

ps 10進	ps 16進	テキスト VRAM文字の属性
0	0	既定の属性
1	1	ハイライト（モノクロのみ）
5	5	ブリンク
7	7	リバース
8 16	8 10	シークレット
30	1E	黒
18 34	12 22	青
17 31	11 1F	赤
19 35	13 23	紫
20 32	14 20	緑

ps 10進	ps 16進	テキスト VRAM文字の属性
22 36	16 24	水色
21 33	15 21	黄色
23 37	17 25	白
40	28	リバース黒
41	29	リバース赤
42	2A	リバース緑
43	2B	リバース黄色
44	2C	リバース青
45	2D	リバース紫
46	2E	リバース水色
47	2F	リバース白

図 2・19 ESC［ps；…；ps m の文字属性

以下でデバイスCONのときと同様に1つずつ説明していきます．

2・7・1 AUXリードデータ（コマンド番号4）

引数(コマンドパケット)の扱いやフローチャートはCONリードデータのときと同じです．CON→AUXに読み換えればよいのです．AUX1バイト入力ルーチンはCONのときと同じように“AUX非破壊入力”をCALLします．ただし，CTRL+PやCTRL+Nが入力されたときの処理はありません．AUX非破壊入力のリターンでバッファに値があればポインタを進めます．

2・7・2 AUX非破壊入力（コマンド番号5）

AUX非破壊入力はCON非破壊入力でINT 18HをINT 19Hにしたことと，特殊キーの処理がなくなったことを除いて同じです．

データ受信（INT 19HのコマンドAH=04H）を実行し，異常終了だったらDONEビットを0にしてリターンします．正常終了だったら入力バッファからポインタを進めずに1バイトのデータを取り出しDONEビットを1（ON）にしてリターンします．**図2・20**のフローチャートを見てください．

バッファリング中のデータがないときもDONEビットを0にしてリターンします．なお，RS-232Cポートからのデータ入力は，INT 0CHのハードウェア割り込みによって常時行われていますので，バッファからデータを読むときにはCLIで割り込みを禁止しています．

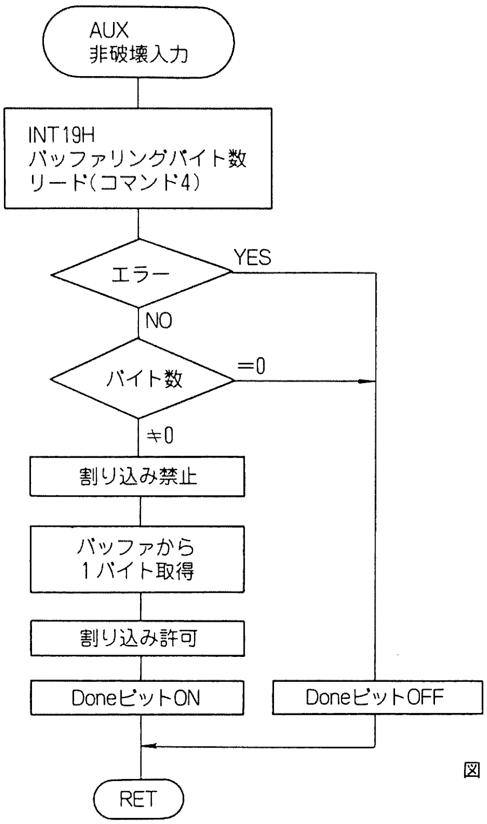


図 2・20 AUX 非破壊入力 1 バイト入力ルーチン・フローチャート

2・7・3 AUX 入力用バッファフラッシュ（コマンド番号 7）

入力用リングバッファのトップ&エンドポインタをバッファ最小アドレスとし，バッファリング文字数カウンタを 0 にします．リングバッファは ROM-BIOS のワークエリアでなく，IO．SYS 内にとられていることを除けば CON のときと同じです．

2・7・4 AUX ライトデータ（コマンド番号 8）

AUX ステータスチェックルーチンを CALL し，送信 Ready となるまで待ちます．送信 Ready となれば

```
MOV AH, 01H ; データ送信
INT 19H
```

を実行します．

2・7・5 AUX 出力用ステータスチェック（コマンド番号 10）

RS-232C ステータスリード

```
MOV AH, 3 ; ステータスリード
INT 19H
```

INT 19H RS-232C ROM-BIOS

本来 INT 19H RS-232C ROM-BIOS は 0〜6 のコマンドを持っていますが，INT 19H の割り込みベクタはセグメント 0，オフセットは 64H〜67H です．IO．SYS の中の処理ルーチンではコマンドは 0〜4 に減っています．コマンドの意味はそれぞれ

0……イニシャライズ			
1……データ送信	CL……送信データ	AH……終了条件	
2……バッファポインタを進める	AH……終了条件		
3……ステータスリード	CL……ステータス	AH……終了条件	
4……データ数	CX……バッファリング中のデータ数		AH……終了条件

です．INT 19H ROM-BIOS とのコマンドとの対応はまったくありません．

AUX（RS-232C）データ受信 INT 0CH

INT 19H のコマンドの中にはデータ受信がありませんでした．RS-232C データ受信についてはハードウェア割り込みを使っています．INT 0CH がそれですが，これもやはり IO．SYS 内に処理ルーチンを持っています（INT0CH の割り込みベクタはセグメント 0，オフセット 30H〜33H です）．

ハードウェア割り込みというのは，ハードウェアからたとえばデータ送信の要求があれば INT 0CH が実行されるようなものです．

ハードウェア割り込みには，このほかに INT 08H〜INT 17H まで，それぞれタイマー，キーボードなどが割り当てられています．

を実行し，送信 Ready なら DONE, Non Busy フラグを 1 にしてリターンします．INT 19H のコマンド 3（ステータスリード）の具体的処理は

```
IN AL, 32H ；μPD8251 ステータスポート
MOV CL, AL
XOR AH, AH
IRET
```

です．μPD8251 のステータスは

bit0……送信 Ready	bit4……オーバーランエラー
bit1……受信 Ready	bit5……フレーミングエラー
bit2……送信バッファエンプティ	bit6……SYNC キャラクタ検出
bit3……パリティエラー	bit7……データセット Reday

となっています．

2・8 デバイス PRN

PRN はプリンタに対応するデバイス名です．したがってコマンド 8(9)，10 それぞれライトデータ(8)，（ライト & ベリファイ(9)），出力用ステータスチェック(10) しかありません（**図 2・10** 参

照).

ROM-BIOS は INT 1AH を使っています.

2・8・1 PRN ライトデータ (コマンド番号 8)

コマンドパケットから指定されたバイト数のデータを PRN 1 バイト出力ルーチンに送るフローは, CON のときと同じです.

では, PRN 1 バイト出力ルーチンはどうなっているかというと, 漢字出力対応プリンタ文字出力コマンド F1H を使っています. すなわち

```
MOV AH, F1H
INT 1AH
```

を実行してプリンタへデータを送っています (図 **2・21**).

2・8・2 PRN 出力用ステータスチェック (コマンド番号 10)

ROM-BIOS のプリンタステータスチェックを利用しています. 図 **2・22** のフローチャートを見て

INT 1AH ROM-BIOS

INT 1AH ROM-BIOS はプリンタとカセットレコーダの BIOS ですが, 現在カセットはサポートされていません. プリンタのコマンドは 10H, 11H, 12H, 30H, F0H, F1H です. INT 1AH も割り込みベクタ (セグメント 0, オフセット 68H～6BH) を書き換えて, IO . SYS の処理ルーチンアドレスを入れています. IO . SYS の処理ルーチンでは AH に入っているコマンド番号が F0H より小さければ, 本来の ROM ルーチンへファージャンプするようになっています. F0H か F1H ならば IO . SYS 内のそれぞれの処理ルーチンへジャンプし, それ以外の番号なら何もせず IRET するようになっています.

それぞれのコマンドに対する機能は

10H……イニシャライズ
11H……データ出力 (シフト JIS 漢字未対応)
12H……ステータスセンス
30H……複数バイトデータ出力 (シフト JIS 漢字未対応)
F0 H……プリンタ BUSY チェック
　　出力 AH　0……non Busy
　　　　　　1……Busy
　　　　　　2……用紙切れ
　　　　　　3……受信不可
F1 H……データ出力 CL…データ (シフト JIS 漢字未対応)

となっています. イニシャライズではコマンド F0H, F1H しか使用していません.

しかし, 10H～30H も使えるようにしてあるので, シフト JIS → JIS 変換をしてほしくなかったら, コマンド 11H か 30H を使用するとよいでしょう. コマンド F1H は送出するデータが漢字の 1 バイト目であれば (図 **2・15** 参照), 1 バイトバッファリングして, つぎに送られてきた 1 バイトのデータと組み合わせて "シフト JIS 漢字コード→ JIS 漢字コード" 変換をし, 漢字シフトコード (セントロニクスプリンタは 1CH, 42H, 1BH, 4BH, 1BH, 02H) を出力しデータを出力します. そのあとにセントロニクスプリンタなら 1BH, 04H を出力します.

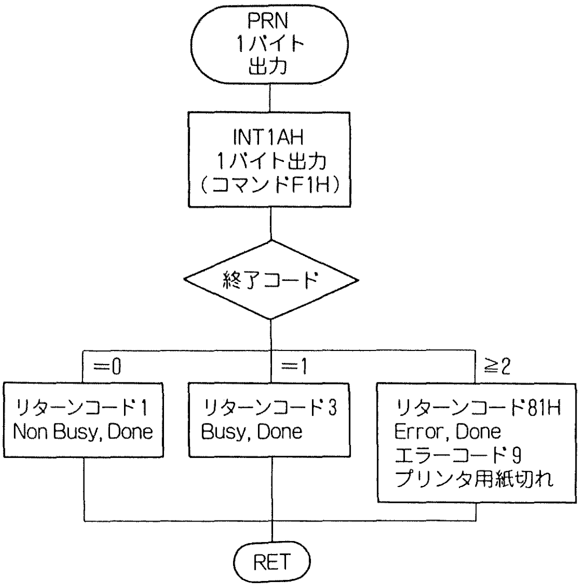


図 2・21 PRN1バイト出力フローチャート

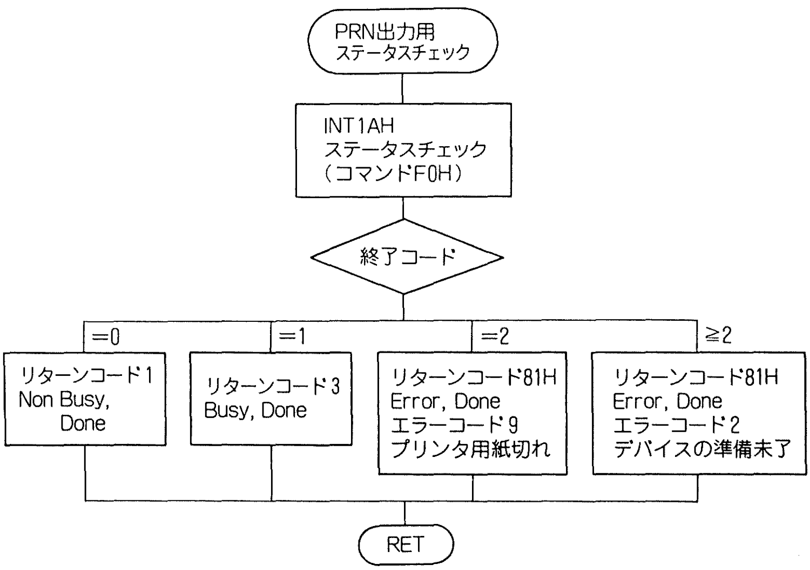


図 2・22 PRN出力用ステータスチェック・フローチャート

ください.

2・9 デバイス CLOCK

CLOCK デバイスとはカレンダ時計 (μPD1990) のことです. このI/Oコントロールコマンドを使って, MSDOS. SYS は日付・時間をセットしたり読み出したりします. CLOCK は CON, AUX, PRN と違って扱うデータは6バイトと決まっています. コマンドは4と8(9)だけです. それぞれ

リードデータ(4)，ライトデータ(8)（ライト & ベリファイ(9)）で日付・時刻の読み出し，セットに対応しています．使用している ROM-BIOS は INT 1CH です．

2・9・1 CLOCK リードデータ（コマンド番号 4）

コマンドパケットから転送アドレスを

```
LES DI, COMPAK+0EH
```

として ES：DI に得ます．日付呼び出しルーチンを CALL すると

AX …… 1980 年 1 月 1 日を基準として何日目かで日付を表わす．
CX ……時刻（CH…時，CL…分）
DX ……時刻（DH…秒，DL…1/100 秒）

をリターンしますので，日付，時・分・秒・1/100 秒の順にバッファに書き込み，DONE ビットをセットしてリターンします．

したがってリードバッファは 6 バイトで以下のようになっています（図 **2・23**）．

さて日付呼び出しルーチンが，まず日付・時刻を呼び出す ROM-BIOS コール

```
XOR AH, AH ；コマンド 0 日付・時刻呼び出し
INT 1CH
```

から始まります．しかし ROM-BIOS の日付・時刻のフォーマットと，I/O コントロールの日付・時刻のフォーマットは違いますから，変換しなければなりません．ROM-BIOS では 1900 年を基準とした年，月，日をデシマル表示で表わします（たとえば 1987 年 3 月 15 日ならば 87H 3XH 15H となります．ここで X の部分は曜日を表わす数字ですが，カレンダタイマ LSI の中で正しく計算されます）が I/O コントロールは 1980 年 1 月 1 日を 0 日目とした日数で表わしますので，日数の変換がいちばん大変です．しかも数字はすべてバイナリ表現となります．くわしくは図 **2・24～26** のフローチャートを見てください．

2・9・2 CLOCK ライトデータ（コマンド番号 8）

リードデータと同じように，転送アドレスが ES：DI に読み込まれていますので

```
MOV SI, DI
ES : LODSW          ；1980 年 1 月 1 日を基準とした日数
CALL 日付ライトルーチン
ES : LODSW          ；時・分
MOV CX, AX
ES : LODSW          ；秒・1/100 秒
MOV DX, AX
CALL 時刻ライトルーチン
```

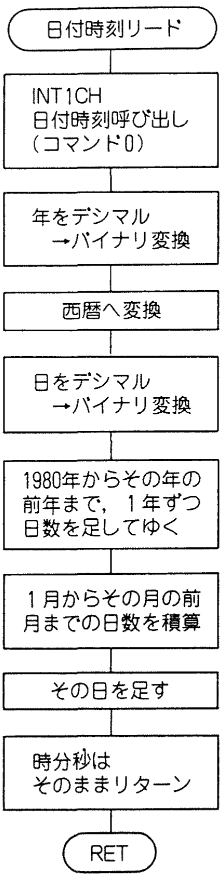
で日付・時刻をセットし DONE ビットを 1 にしてリターンします．バッファの形式は図 **2・23** と同じです．

日付・時刻ライトルーチンは図 **2・28，29** を見てください．

+0	+1	+2	+3	+4	+5
日	付	分	時	$\frac{1}{100}$秒	秒

1980年1月1日を1日目とした日数　(注)数字はすべてバイナリ型

図 2・23　CLOCK リード（ライト）バッファ



月の日数テーブル

31	*	31	30	31	30	31	31	30	31	30	31
↑1月	↑2月										↑12月

2月はその年の日数計算ルーチン(図2・25)でセットされる

図 2・24　日付・時刻リードルーチン

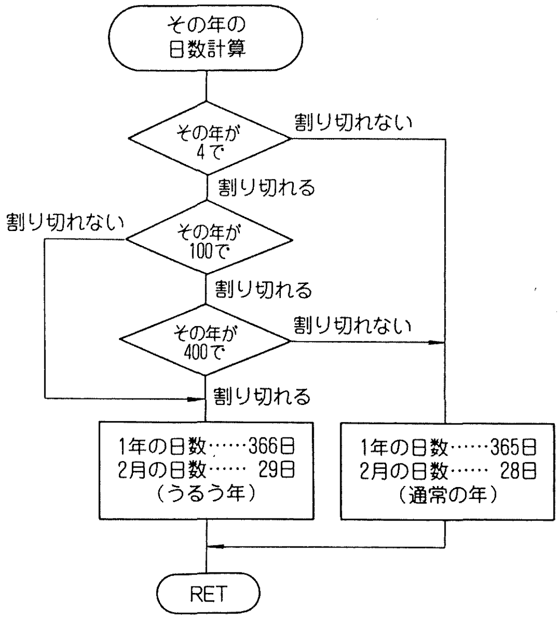


図 2・25　うるう年計算ルーチン

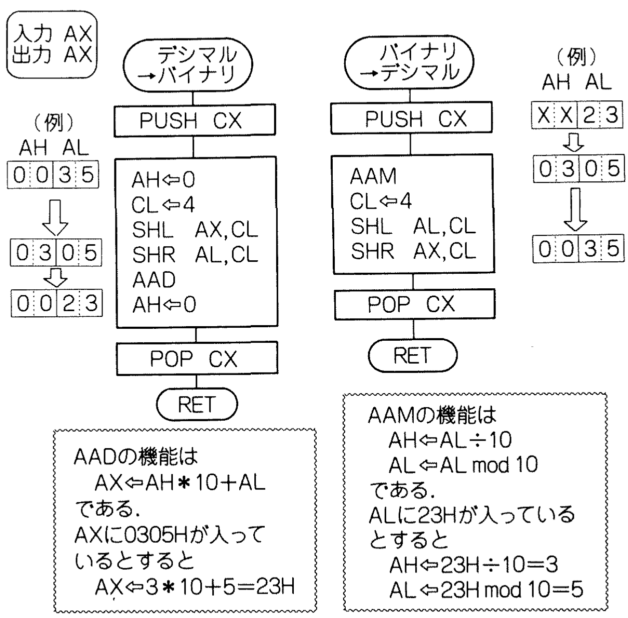


図 2・26　デシマル－バイナリ相互変換ルーチン

INT 1CH カレンダ・インターバルタイマ ROM-BIOS

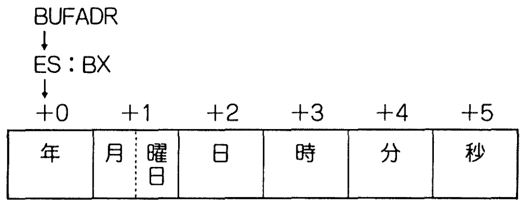
INT 1CH はカレンダ時計，インターバルタイマに関する ROM-BIOS で以下のコマンドがあります．

0……日付・時刻呼び出し

1……日付・時刻セット

2……インターバルタイマ割り込みのセット

コマンド 2 はタイマ割り込みで CX…インターバルタイマ（×10msec），ES：BX…インターバルタイム経過後の処理エントリアドレスです．CLOCK で関係するのはコマンド 0 と 1 だけで，日付・時刻バッファ先頭アドレスを ES：BX にセットしてコマンドを発行します．バッファの形式は図 **2・27** のようになっています．



数字はそれぞれデシマル表現で（例）32なら32Hとなる．
年は1900年を基準としている．（例）1987年なら87Hとなる．
曜日は下表に対応している．

0	1	2	3	4	5	6
月	火	水	木	金	土	日

図 **2・27** ROM-BIOS INT 1CH の日付・時刻バッファ

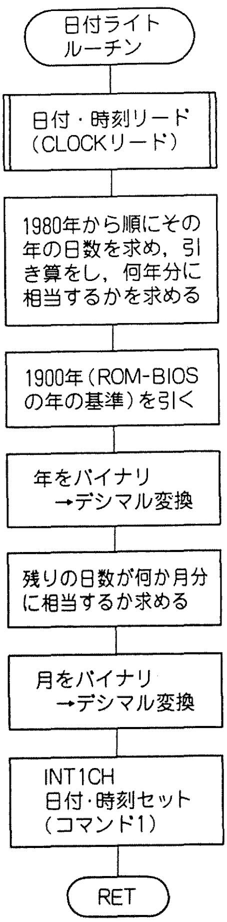
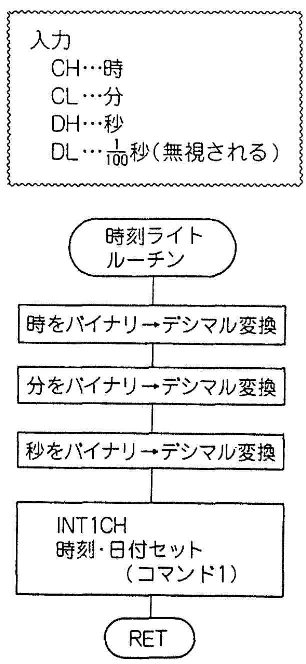


図 **2・28** 日付ライトルーチン

図 **2・29** 時刻ライトルーチン



2・10 ディスクデバイス

デバイスは**図 2・10** を見てもらえばわかるように，初期化(0)，メディアチェック(1)，BPB 作成(2)，リードデータ(4)，ライトデータ(8)，ライト & ベリファイ(9) をサポートしています．ディスクのほうも 8 インチ・フロッピーディスク(5 インチ 2HD を含む)，5 インチ 2DD，ハードディスクと多様なディスクドライブをサポートしています．

2・10・1 ディスク初期化（コマンド番号 0）

ハードウェアの初期化はシステム立ち上げのとき ROM-BIOS 内のイニシャライズルーチンで行っているので，ここではソフト的な初期化のことです．

コマンドパケットへのリードデータ値は**図 2・8** のようになります．コマンドパケットの先頭アドレスを COMPAK とすると，COMPAK＋0DH は“デバイスドライバがサポートしている BPB 数”です．この数は IO．SYS のワークエリアの PDA バッファや接続されているディスクドライブ数などから求めます（**図 2・1** 参照）．

COMPAK＋0E～11H の“デバイスドライバの RAM 上の最終アドレス”は IO．SYS の標準デバイスドライバであることからセットされません．COMPAK＋12H～15H は“BPB へのポインタの表アドレス”です．BPB 表と BPB へのポインタの表は IO．SYS 内にあらかじめ用意されていますので，この表の先頭アドレスをセットするだけです．以上の事項をセットしたら DONE ビットを 1 にしてリードデータします．

図 2・30，31 に BPB 表と BPB へのポインタの表の実例を示します．

アドレスはバージョンによって違いますので，各ポインタのリンク状態を BPB 作成時の参考程度に見てください．

```
DEBUG
-D60:89C2,8A64                                                        ①
0060:89C0        [3E 8A] [3E 8A] [E3 89]-[E3 89] [3E 8A] [3E 8A] [3E 8A] ↑   >.>.c.c.>.>.>.
0060:89D0  [3E 8A] [4B 8A] [58 8A] 80 00-04 01 00 02 44 00 D2 07      ↓  >.K.X.......D.R.
0060:89E0  00 06 00 00 04 01 01 00-02 C0 00 D0 04 01 02 00            ②  .........@.P....
0060:89F0  [80 00] [04] [04 00] [02] [44 00]-[D2 07] [02] [06 00] [00 02] [01]    ......D.R.......
0060:8A00  [01 00] [02] [40 00] [40 01] [00]-[01 00] 00 02 01 01 00 02        ...@.@..........
0060:8A10  40 00 68 01 01 02 00 00-02 02 01 00 02 70 00 80               @.h..........p..
0060:8A20  02 02 01 00 00 02 02 01-00 02 70 00 D0 02 03 02               ..........p.P...
0060:8A30  00 [00 02] [02] [01 00] [02] [70-00] [00 05] [04] [02 00] [00 02]    .......p........
0060:8A40  [02] [01 00] [02] [70 00] [A0 05]-[05] [03 00] [00 00] 00 00 00      ....p. .........
0060:8A50  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00               ................
0060:8A60  00 00 00 00 00                                                .....
-Q
```

① BPB へのポインタの表

② BPB の表

アドレスはバージョンによって異なる．

図 2・30 BPB の表と BPB へのポインタの表の実例

	ドライブ種別		5インチ						ハードディスク	8インチ		
			1D		2D		2DD			2D		
	図2・30の実例のアドレス		89FD	8A0A	8A17	8A24	8A31	8A3E	8A4B	89D6	89E3	89F0
オフセット	内　容	占有バイト	8セクタ/トラック	9セクタ/トラック	8セクタ/トラック	9セクタ/トラック	8セクタ/トラック	9セクタ/トラック				
+0	バイト/セクタ	2	512	512	512	512	512	512	0	128	1024	128
+2	セクタ/アロケーション	1	1	1	2	2	2	2	0	4	1	4
+3	予備のセクタ	2	1	1	1	1	1	1	0	1	1	4
+5	FATの数	1	2	2	2	2	2	2	0	2	2	2
+6	ディレクトリの数	2	64	64	112	112	112	112	0	68	192	68
+8	論理イメージのセクタ数	2	320	360	640	720	1280	1440	0	2002	1232	2002
+A	メディアディスクリプタ	1	0	1	2	3	4	5	0	0	1	2
+B	FATセクタ数	2	1	2	1	2	2	3	0	6	2	6

(注) 8″フロッピーは5″−2HDフロッピーも含む　(注) ハードディスクを接続していないので内容が0だった
(注) アドレスはバージョンによって異なる

図 2・31 「図 2・30」を表にまとめたもの (BPBの内容)

2・10・2 ディスクメディアチェック (コマンド番号1)

コマンドパケットの +1バイト目の示す論理装置コード (図2・8参照) にドライブ番号 (A→0, B→1, C→2, …) が入っているとして，そのコードが示すドライブに対してディスクが交換されたかどうかをチェックします[注]．大まかな流れは図2・35のようになります．

論理装置コード→PDA変換は図2・1に示したワークエリアのPDAテーブルを使用します．ドライブAに対するPDAから順番に格納されています．テーブルの内容が00HのドライブにはIO.SYSの管理しているディスクが接続されていないことを表わします．論理装置コード→PDAの変換は

```
MOV BX, テーブルアドレス
XLAT
```

によって行う (ALレジスタにPDAを得る) ことができます．

つぎに，得られたPDAに対するディスクが交換されたかどうかのチェックを行います．このためには，PDAの上位4ビットによってディスクの種類に分類し，さらにFATのIDやステータスセンスなどによってディスクの交換をチェックします (図2・36参照)．

まずいちばん簡単なのはハードディスクで，ディスクの交換が不可能なのでつねに“ディスクは交換されていない”を示す1をAHにセットしてリターンします．

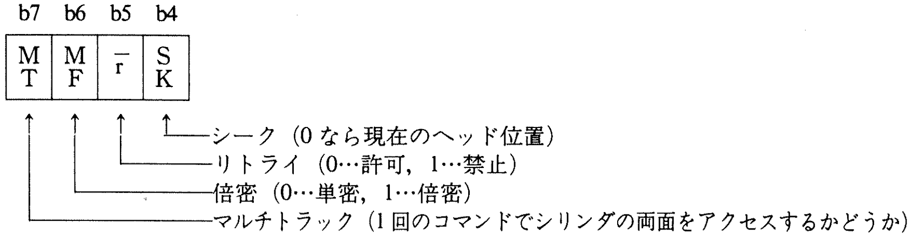
つぎに5インチ・フロッピーディスクですが，まず

```
MOV     BP, 512バイトのバッファ先頭アドレス
MOV     CX, 200H          ; 512バイト/セクタ, シリンダ0
```

[注] 図2・8ではメディアディスクリプタをセットするようになっていますが論理装置コードが使われます．

INT 1BH ディスク ROM-BIOS

ディスク用の ROM-BIOS として INT 1BH があります．コマンド番号は 0～F (10H) で AH レジスタにセットして INT 1BH コールを行いますが，AH レジスタの上位 4 ビットは



コマンド	機能	引数使用レジスタ AH	AL	BX	CH	CL	DH	DL	ES : BP
0	AH の b4=1 (SK ビット=1) と組みにしてシークコマンド	○	○	–	–	○	–	–	–
1	ベリファイ	○	○	○	○	○	○	○	○
2	リードダイアグノスティック	○	○	○	○	○	○	○	○
3	初期化 (5″-1D，2D は入力に AL 使用)	○	– (○)	–	–	–	–	–	–
4	ステータスセンス	○	○	–	–	–	–	–	–
5	ライトデータ	○	○	○	○	○	○	○	○
6	リードデータ	○	○	○	○	○	○	○	○
7	リキャリブレイト	○	○	–	–	–	–	–	–
8	代替トラックの指定	○	○	○	◎	◎	–	◎	○
9	ライトデリーテッドデータ	○	○	○	○	○	○	○	○
A	リード ID	○	○	–	–	○	○	–	–
B	フォーマット Bad Track (不良トラックの代替トラックへの割り付け)	○	○	○ BX←4	◎	◎	–	◎	○
C	リードデリーテッドデータ	○	○	○	○	○	○	○	○
D	フォーマットドライブ (5″-1D, 2D) フォーマットトラック (8″，5″-2DD) フォーマットトラック/ドライブ (ハードディスク)	○	○	○	○	○	○	○	○
E	動作モード (両面/片面) 設定 (5″-1D, 2D)	○	○	–	–	–	–	–	–
F	リトラクト：アームの不使用シリンダへの移動 (ハードディスク)	○	○	–	–	–	–	–	–

(注) 引数使用レジスタで◎は図 2・33 の一般的な使い方になっていないもの．

図 2・32 INT 1BH ディスク ROM-BIOS コマンド一覧

レジスタ	意　　味
AH	コマンド（図2・32）
AL	PDA
BX	データバッファ(ES:BP)のバイト数
CH	セクタ長($128*2^{CH}$バイト, CH=0, 1, 2, …, 5)
CL	シリンダ番号
DH	サーフェス番号(表…0, 裏…1)
DL	セクタ番号
ES:BP	データバッファアドレス

(例外)

・5″-1D, 2D の初期化(コマンド3)

AL…[0|1|0|1| | | |] UA (Unit Address)
- bit 0, 1, 2, 3: 0にすると対応ドライブが片面、1にすると対応するドライブが両面モードになる.

・ハードディスクの代替トラックの指定(コマンド8)

BX⇐4, CX…代替トラックのシリンダ番号
DH…　〃　ヘッド番号
DL…0

・ハードディスクのフォーマット Bad Track(コマンドB)

BX⇐4, CX…不良トラックのヘッド番号, DL⇐0

・フォーマットトラック/ドライブ

DL…データパターン(フロッピー), 0(ハードディスク)

図 2・33 INT 1BH ディスク ROM-BIOS の一般的引数

エラー番号	意　　味
08	コレクテッドデータ検出
10	DDAM を検出した
20	バッファ先頭アドレス（ES：BP）が奇数またはバッファがセグメント境界を含む
30	シリンダを越えた書き込み
40	デバイス異常，リキャリブレイト失敗
50	書き込み時間超過
60	ドライブの準備ができていない
70	ライトプロテクトされている
78	指定セクタがレンジオーバー
80	原因不明のエラー
88	代替トラックを直接アクセスした
90	タイムアウト
A0	ID 部の CRC エラー検出
B8	データ部でエラー検出
C0	指定セクタが存在しない
C8	シークエラー
D8	代替トラックが読めない
E0	指定セクタがなく，正常な ID もない

(注) エラー番号は 16 進

○ 5″-1D, 2D フロッピーのときキャリーフラグ=0 なら

AH の bit0… { 0 動作モードが片面 / 1　〃　両面 }

bit1… { 0 片面装置 / 1 両面〃 }

図 2・34 INT 1BH ディスク ROM-BIOS キャリーフラグ=1 でリターン時のエラーコード

という意味を持っています．コマンドを**図 2・32** に示しますが，コマンド 0 は AH の bit4＝1（シーク）と組みになってシークコマンドを形成します．引数はレジスタ渡しで 2～3 の例外を除いて**図 2・33** のようになっています．リードデータ値はキャリーフラグが 1 のときエラーで，AH に**図 2・34** のようなエラー番号が返されます．

ここで PDA＝Physical Device Address で PDA は下位 4 ビットがドライブ A → 0，B → 1……と対応しており，上位 4 ビットはディスクの種類によって

5……5 インチ 1D・2D フロッピーディスク
7……5 インチ 2DD フロッピーディスク
8……ハードディスク
9……8 インチ 2D（5 インチ 2HD）フロッピーディスク

と対応しています．

```
MOV     DX, 0002H      ; サーフェス 0, セクタ 2
MOV     BX, 200H       ; 読み込むデータバイト数 512
MOV     AL, PDA の下位 4 ビット
OR      AL, 70H
MOV     AH, D6H        ; データリード
INT     1BH
```

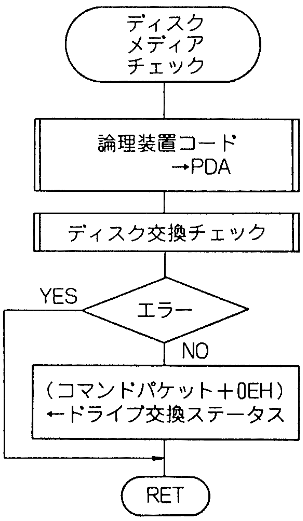


図 2・35　ディスクメディアチェック概要フローチャート

DA	70H						80H	90H		
ディスクタイプ	5″-1D		5″-2D		5″-2DD		ハードディスク	8″-1D	8″-2D	8″-1D
FAT ID	FE	FC	FF	FD	FB	F9	—	FE	FE	FD
セクタ/トラック	8 セクタ/トラック	9 セクタ/トラック	8 セクタ/トラック	9 セクタ/トラック	8 セクタ/トラック	9 セクタ/トラック	—	—	—	—
メディアディスクリプタ	0	1	2	3	4	5	0	0	1	2

図 2・36　PDA の DA，FAT の ID によるディスクの分類　　DA（Device Address）

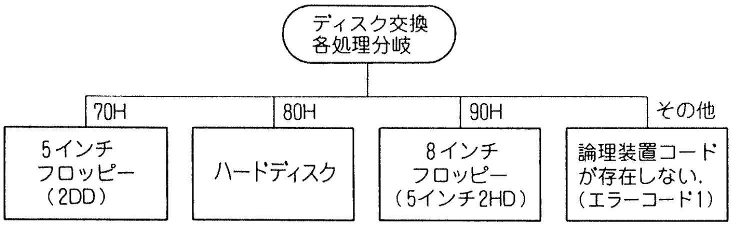


図 2・37 ディスク交換チェックルーチンの PDA の DA による分類

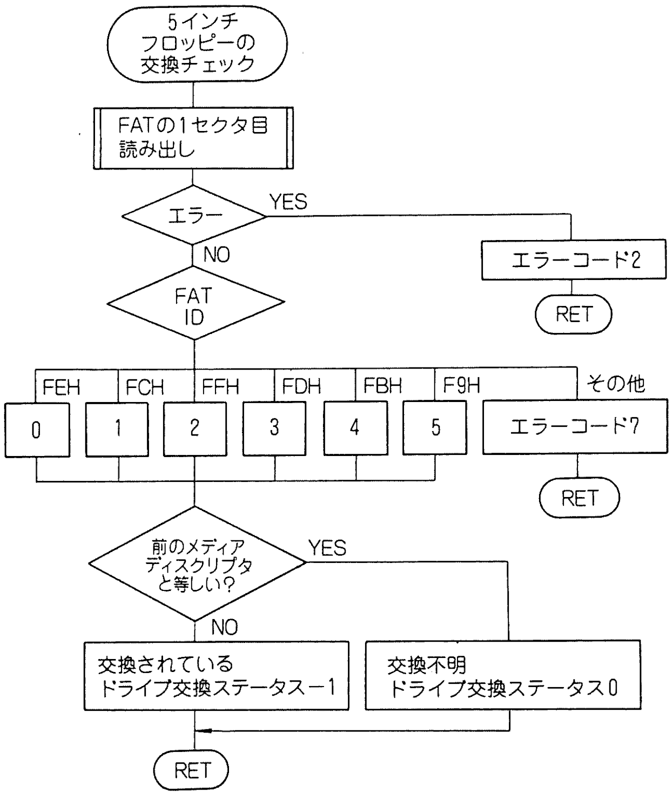


図 2・38 5インチ・フロッピーの交換チェック

によって FAT を読み込みます. INT 1BH の結果エラー番号 60H"ドライブの準備ができていない"ならばキャリーフラグを 1, エラーコードを**図 2・37** のようにしてエラー終了します. もしそれ以外のエラーならば INT 1BH を 10 回リトライします.

さて, このようにしてメディアディスクリプタに変換し, PDA の下位 4 ビットに対応するメディアディスクリプタ (**図 2・1**) と比較し, 変化していなければ「交換不明 (0)」と判断し, 異なっていれば「交換されている (−1)」と判断します(**図 2・1** の (10) 参照).

(10)　は 5 インチ・フロッピーディスクの装置 0

(10)＋1 は同じく装置 1

(10)＋3 は装置 3

のメディアディスクリプタ (**図 2・36** 参照) を格納しています. もしディスクが交換されていればこのメディアディスクリプタが変化するはずです.

いずれにしても 5 インチ 2DD・フロッピーでは「交換されていない」というはっきりした結論は

出せません.

つぎに，8インチ・フロッピーディスクの交換のチェックですが，8インチは図**2・36**を見てもらえばわかるように，FAT IDがFEHと同じものがあるのでちょっとやっかいです．FAT IDが同じものが8"-1Dと8"-2Dなので片面・両面を区別して，それからFAT IDによるチェックを行っています．FAT IDによるチェックでは交換されていないと断定することはできませんが，8"-FDではレザルトステータスのST0のbit6＝bit7＝0でなければ交換されていないと断定することができます.

2・10・3　ディスクBPB作成（コマンド番号2）

IO.SYS内に持っている5"-FD, 8"-FD, HDに関するBPB表の中から論理装置コードで示されるドライブ番号に対応するディスクのBPBの先頭アドレスをコマンドパケットアドレス＋12～15Hに返します（12～13H…オフセット，14～15H…セグメント）.

IO.SYS内のBPB表は図**2・30**，**31**のようになっていますので，図**2・40**のフローチャートのよ

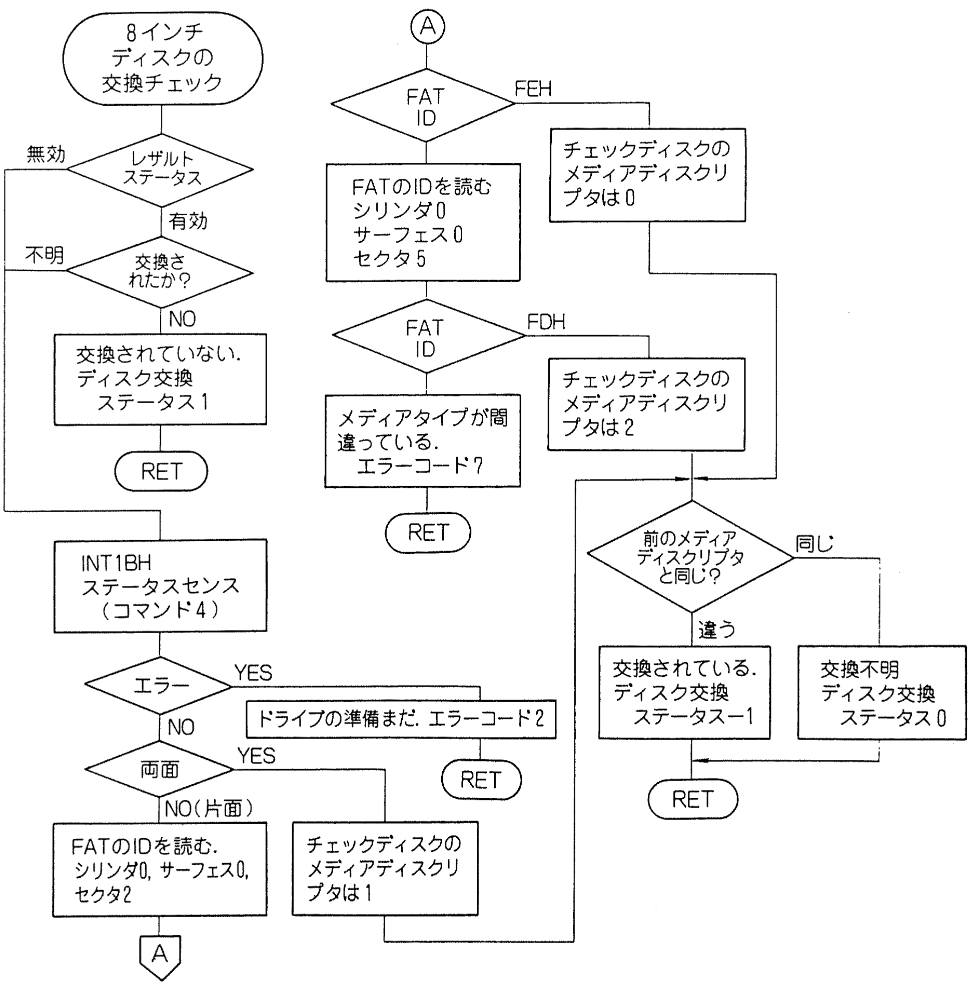


図 **2・39　8インチ・フロッピーの交換チェック**

8”-FD レザルトステータス

セグメント 0, オフセット 564H～583H の 32 バイトは 8”-FD のレザルトステータスエリアとなっています. 8”-FD は 4 台まで接続可能なので, 1 ドライブに 8 バイト×4 台＝32 バイトの領域が確保されているのです. 各バイトは

＋1	ST0	…… ステータス 0	
＋2	ST1	…… ステータス 1	
＋3	ST2	…… ステータス 2	
＋4	C	…… シリンダ	(ディスク ID 部の CHRN のこと)
＋5	H	…… ヘッド	
＋6	R	…… セクタ番号	
＋7	N	…… セクタ長	
＋8	NCN		

となっており, 以下ドライブ 3 までこの繰り返しとなっています.
くわしくは「PC-TechKnow 9800 (システムソフト刊)」P. 435 を参照してください.

うにして対応する BPB のアドレスを求めています.

BPB 作成というと, BPB をあらたにメモリ上に作成するように聞こえますが, フローチャートで

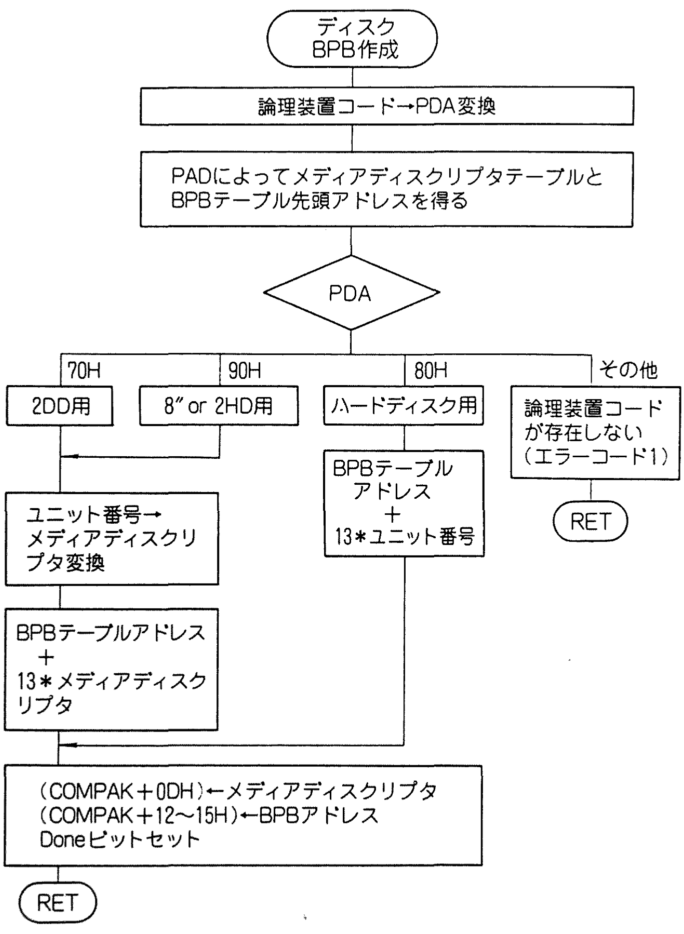


図 2・40 ディスク BPB 作成フローチャート

示したように, IO . SYS 内にシステム起動時から用意されている BPB の中から要求されたディスクに対応するものの先頭アドレスをリターンするだけです.

したがって, BPB 作成のコマンドを発行したほう (MSDOS . SYS) では BPB のための領域をあらたにメモリ上にとることはしません. ただし, BPB は IO . SYS のメモリ領域にあるので, もっと使いやすい形 (DPB) にして MSDOS . SYS のメモリ領域にコピーします.

ここでメモリ領域というのは, IO. SYS, MSDOS. SYS それぞれに異なったセグメントベースを持っているので, それぞれのセグメントベースのメモリ上へということです.

論理装置コードというのは, ここではドライブ番号 (ドライブ A, B, C・・・) のことです. ドライブ番号をディスク BIOS が扱う PDA というかたちに変換するには, IO. SYS のオフセットとして図 **2・1** の (9) の「論理装置コード (ドライブ番号) → PDA 対応表」を使います. PDA の上位 4 ビットはディスクメディアによって 7 (5"-2DD), 8 (ハードディスク), 9 (8"-2D, 5"-2HD) と分類できますので, あとはそれぞれに対応する BPB テーブルのアドレスを求めてリターンするのです. PDA については 65 ページのコラム (INT1BT ディスク ROM-BIOS) を参照してください.

メディアディスクリプタは, ここでは同じディスクメディアの中で何番目のディスクかということです. すなわち, ROM-BIOS では同じ種類のディスクドライブは一連の番号をつけて管理しますが, その番号のことです (PDA の下位 4 ビット).

2・10・4　ディスクリードデータ (コマンド番号 4)
ディスクライトデータ (コマンド番号 8)
ディスクライト & ベリファイ (コマンド番号 9)

リードデータとライトデータは, それぞれディスク→メモリ, メモリ→ディスクとデータの転送方向が違うだけで, データバッファやデータのセクタ, トラックなどの指定は同じなので, ひとま

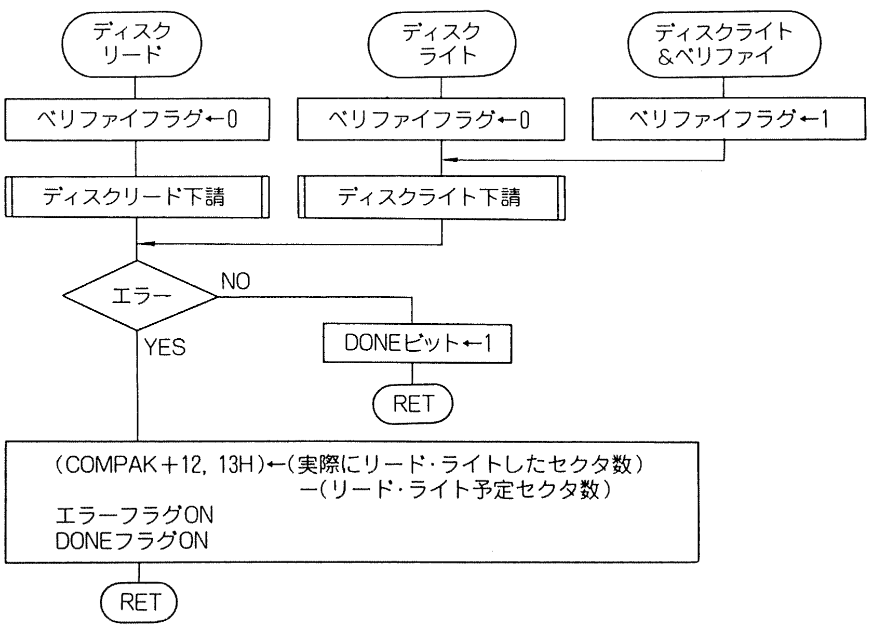


図 2・41　ディスクリード/ライト/ベリファイルーチン概要

とめに考えることができます．また，ライト＆ベリファイはライトしたあとにベリファイを行うかどうかの違いだけなので，ディスクIO.SYSの中のベリファイフラグによって対応しています．

これら3つの処理の概要は**図2・41**のフローチャートのようになっています．

第3章

MSDOS.SYSの仕組みと解析フローチャート

この章の方針

前章はIO.SYS内のおもにデバイスドライバの説明であり，外部コマンドから直接アクセスすることはあまりなく，自分でデバイスドライバを作るときくらいしか役立つ情報ではありませんでしたが，本章で解析してゆくMSDOS.SYSは大部分がINT 21Hファンクションコールの処理ルーチンであり，外部コマンドで直接呼び出すことのもっとも多い部分です．したがってINT 21Hファンクションコールの MSDOS.SYS内での処理の様子を知ることは，外部コマンドを自作する上で大きな助けとなることでしょう．

MSDOS.SYSではこのINT 21Hファンクションコールのほか，以下のような機能を持っています．

○　INT 20H　　プログラム終了
○　INT 25H，26H　　物理セクタ　リード/ライト
○　INT 27H　　プログラム常駐終了
○　INT　0　　除算エラー（0で割った）処理

なお，コンソール入出力時のテンプレート機能はINT 21Hファンクションコールのファンクション0AH，バッファードキーボード入力の中の機能です．

以下で順番にこれらの機能の解析を進めてゆきますが，まずMSDOS.SYSの基本的な要素(たとえば，"内部FCB"，"ディスクバッファ"など)についての説明を行ってから，INT 21H各ファンクションコールについての説明をすることにしましょう．

MSDOS.SYSがハードに関する処理をするときは，IO.SYSに対して"I/Oリクエストコール"を発することによって行いますので，前章で説明したIO.SYSのデバイスドライバをコールする側から見ることになり，IO.SYSに対してもいっそう理解が得られることになるでしょう．

3・1　INT 21Hファンクションコールの未公開コマンド

INT 21Hファンクションコールは，AHレジスタにファンクション番号を入れて実行するようになっています．そして引数はレジスタを通して渡されます．ファンクション番号は0～58Hのほか，OEM拡張用にF8H～FFHが割り当てられています．0～58Hのほうは1つの番号に1つの機能が割り当てられていますが，公開されている機能はところどころ番号が歯抜けになっています．これらの番号の中にはなにも処理せずすぐにリターンするものもありますが，システム予約という形でなんらかの処理をする番号もあります．いわゆる"隠しコマンド"です．これらのコマンドの中にはVer 2. xxだけしか使えないものもありますが，Ver 3. xxにも使えるものもいくつか存在します．たとえば，ファンクションコール番号32Hは公開されていませんが，メディアチェックしてDPBを更新し，DPBアドレスをDS：BXにリターンする機能があり，Ver 3. xxでも使用可能です．

またこれらのコマンドの中には，MSDOS.SYSの解析上重要なパラメータテーブルの格納アドレスを返すものもあるので，ここではやや変則的ではありますが，この未公開コマンドの使い方から説明してゆきましょう．

<table>
<tr><th>ファンクション番号</th><th colspan="3">機　　　　能</th></tr>
<tr><td>18H</td><td colspan="3">NOP</td></tr>
<tr><td>1BH</td><td colspan="3">カレントドライブ（DL=0）を指定してファンクション1CHを実行</td></tr>
<tr><td>1CH</td><td colspan="3">DLで指定されるドライブのメディアチェック&DPB更新
AL …セクタ/クラスタ
CX …バイト/セクタ
DX …データエリア総クラスター1
DX：BX…DPB先頭アドレス+16H…メディアディスクリプタバイトの位置</td></tr>
<tr><td>1DH
1EH</td><td colspan="3">NOP</td></tr>
<tr><td>1FH</td><td colspan="3">カレントドライブを指定してファンクション32Hを実行</td></tr>
<tr><td>20H</td><td colspan="3">NOP</td></tr>
<tr><td>26H</td><td colspan="3">DS：DX～に新しいPSPを作る．呼ばれたプログラムのPSPから．</td></tr>
<tr><td>32H</td><td colspan="3">DLで指定されるドライブのDPBアドレスの取得．DS：BXにリターンされる．</td></tr>
<tr><td>34H</td><td colspan="3">ES：BXにINT21H処理中を表わすフラグの格納アドレスを得る．</td></tr>
<tr><td>37H</td><td colspan="2">システムパラメータ・セパレータ（デフォールト'/'2FH）の
AL=0…リード，1…ライト
デバイス名許可フラグのAL=0…リード，2…ライト</td><td>DLにセット
リターン</td></tr>
<tr><td>50H</td><td colspan="3">カレントPSPアドレスのセット．BXにセグメントアドレス．</td></tr>
<tr><td>51H</td><td colspan="3">カレントPSPアドレスのリード．BXにセグメントアドレス．</td></tr>
<tr><td>52H</td><td colspan="3">システムパラメータ・テーブル・アドレスの取得．ES：BXにそのアドレス．
ES：BX　+0～3 DPB TOP　+4～+7 内部FCB　+8～+B CLOCKデバイスヘッダ
+C～+F デフォールトのデバイスヘッダTOP</td></tr>
<tr><td>53H</td><td colspan="3">BPB（IO. SYS, デバイスドライバ）の内容をDPB（MSDOS. SYS）形式に変換
DS：SI → ES：BP</td></tr>
<tr><td>55H</td><td colspan="3">新しいPSPをカレントPSPから作り，全ファイルクローズする．
セグメントDXに作る．</td></tr>
<tr><td>58H</td><td>MCB割当て法のリード/ライト．
AL…0 リード　AXにリターン値
1 ライト　BXの値をセット</td><td colspan="2">割当て法
0…最下位のMCBを割り当てる（デフォールト）．
1…フリーMCB中最小のものを割り当てる．
2…最上位アドレスにMCBを割り当てる．</td></tr>
<tr><td>F8H</td><td colspan="2">拡張用コマンド処理アドレスセット．アドレスはDS：DX．</td><td rowspan="2">OEM
拡張用
（現在未使用）</td></tr>
<tr><td>F9H
≀
FFH</td><td colspan="2">F8Hでセットされたアドレスへファージャンプする．</td></tr>
</table>

（注）NOPはAL=0としてリターンするだけ．

図3・1　未公開INT 21Hファンクションコール

3・1・1　INT 21H 未公開ファンクションコールの使いかた

未公開ファンクションコールをまとめると**図 3・1** のようになります．実に 14 個(NOP と拡張用の F9H～FFH を含まず)もあります．これらのファンクションコールを入れると番号は 0～58H まで連続したものとなります（Ver 3. xx では 59H からいくつか拡張されている)．

以下にこれらのファンクションコールの解析結果を述べます．

各ファンクションコールの説明は以下のような形式によっています．

ファンクション番号――ファンクションの簡単な見出し――

呼び出し法

AH　ファンクションコール番号
BX　引数はレジスタに入れてコールするので，引数の意味の説明
CX　必要なレジスタだけを列記する．
DX

リターン値

AL　リターン値はレジスタに入れられてもどるので
BX　どういう値がレジスタに入れられてもどるのかの説明
CX　必要なレジスタだけを列記する
DX

説明

そのファンクションに対するくわしい説明

(1)　ファンクション 1BH，1CH
――セクタ/クラスタ，バイト/セクタ　データエリア総クラスタ取得――

呼び出し法

AH　1BH または　1CH
DL　ドライブ番号（ファンクション番号 1CH のとき，1BH はカレントドライブになる）

リターン値

AL　セクタ/クラスタ
CX　バイト/セクタ
DX　データエリア総クラスター1
DS：BX　DPB アドレス＋16H

説明

ファンクション 1CH は，DL で指定されるドライブの

セクタ/クラスタ　→　AL
バイト/セクタ　→　CX
データエリア総クラスター1　→　DX

をリターンします．このついでに DS：BX に DL で指定されたドライブに対応する DPB アドレス＋16H をリターンします．DPB アドレス＋16H といえば，メディアディスクリプタバイトの位置

です．ただし，DPB アドレスを得たいのであればファンクション 32H のほうが確実です．このファンクションは AL，CX，DX にリターンされるもののほうがメインなのですから．

ここで，DL の値とドライブの対応は

DL	=0	カレントドライブ
	1	ドライブ A
	2	ドライブ B
	3	ドライブ C
	⋮	⋮
	10H	ドライブ P

となっています．

ファンクション 1BH のほうはカレントドライブの

セクタ/クラスタ	→	AL
バイト/セクタ	→	CX
データエリア総クラスター1	→	DX

をリターンします．すなわち，DL ← 0 としてファンクション 1CH をしたのと同じ処理になります．ただし，これらのファンクションコールとほぼ等価なものが公開ファンクションコール(番号 36H)にあるので，これらを使うよりは公開されているファンクションコール 36H を使ったほうがはるかによいでしょう．つぎから述べるファンクションコールについては，公開ファンクションコールでは実現できないものばかりです．

(2)　ファンクション 1FH，32H　――DPB アドレスの取得――

呼び出し法

AH	1FH，32H
DL	ドライブ番号（ファンクション 32H のときのみ）

リターン値

DS：BX　DPB の先頭アドレス

説明

ファンクション 1FH は前節の 1BH と 1CH の関係のように，32H の機能をカレントドライブに対して行うようにしたものです．そこでファンクション 32H の機能はというと，DL で指定されたドライブの DPB アドレスを DS：BX にリターンすることです．このときメディアチェックを実行しディスクが入っていなければその旨報告し，交換されていれば DPB 更新をします．DPB 更新というのは，一口でいえばそのディスクの BPB を I/O リクエストコールで取り出しファンクション 53H を使って BPB から DPB を作り出すことです．BPB の内容については IO.SYS の解析の章の図 **2・31** (p. 64)に示しましたが，DPB については図 **3・2** に示します．

DPB は内容的には BPB とほぼ同じようなものですが，ディスクアクセスなどの処理を迅速に行うために，DPB のほうが内容がくわしくなり全体の大きさも大きくなっていることがわかります．MSDOS.SYS はこの DPB を駆使してディスクを管理しています．DPB の中にはそのドライブの対応するデバイスヘッダへのベクタやカレントディレクトリのパス名など興味深いものが含まれています．ただし，カレントドライブ・ディレクトリのパス名は Ver 3.xx では DPB からはなくなっています．

DPB	
+0	ドライブ番号
+1	論理装置コード
+2, 3	バイト数/セクタ
+4	(セクタ/クラスタ)−1＝2^n-1
+5	nの値 ←(シフトカウンタ)— $\log_2$（セクタ/クラスタ）
+6, 7	予備のセクタ数
+8	FAT数
+9, A	ディレクトリ総数
+B, C	データエリアTOPセクタ
+D, E	データエリア総クラスタ
+F	セクタ/FAT
+10, 11	ディレクトリTOPセクタ
+12〜15	デバイスヘッダへのベクタ
+16	メディアディスクリプタ
+17	ディスク交換フラグ
+18〜1B	つぎのDPBへのポインタ
+1C, 1D	カレントディレクトリクラスタ
+1E〜5D	カレントディレクトリへのパス名（64バイト）

図3・2　DPB（Device Parameter Block）の内容

(3)　ファンクション26H，55H——PSPの作成——

呼び出し法

AH　26H，55H

リターン値

DX　新しくPSPを作りたいセグメント

ファンクション26H

CS：0〜FFHのPSPをもとにして新しいPSPを作る

ファンクション55H

カレントPSPをもとにして新しいPSPを作る

説明

両ファンクションともDXで示されるセグメントにPSPを作成することは同じですが，26Hのほうは INT 21Hを実行したプログラムのコードセグメントCS上のオフセット0〜FFHにあるPSPをもとにして，55Hのほうはカレント PSP（ファンクション50H，51H参照）をもとにしてPSPを作成します．したがって，26HのほうはCOM型のプログラムにしか使えません．

具体的にはPSP作成方法は，まずもととなるPSPの0〜FFHの内容をDX：0000H〜に単純に転送します．つぎに，INT 22H，23H，24Hの割り込みベクタ（すなわち，セグメント0のオフセット88H〜93H）の内容をDX：000AH〜0015Hへ転送します．そして，DX：0006，7HにPSPセグメントからRAM上限までのバイト数（FFF0Hより大きいときはFFF0H），DX：0008，9Hに

000CH−((RAM上限のセグメント+1)−DX)　………………①

（↑ RAM上限のセグメント+1：PSPの［0002，3H］）

を，DX：0000，1H に INT 20H に当たる CDH，20H を，DX：0005H に CALLF に当たる 9AH を，DX：0050H～52H に INT 21H　RETF に当たる CDH，21H，CBH を格納します．

ファンクション 26H のほうはこれだけでリターンするのですが，55H のほうは PSP 内のファイルハンドルに関するシステムワークエリアをコピーします．

ところで PSP のオフセット 5～9H には，得態の知れぬ計算結果が格納されていますが，これはいったい何なのでしょう？　オフセット 8，9H に格納するための計算（上述式①）の 000CH というセグメントは 000C：0000H⟶0000：00C0H と，PSP のオフセット 5H との間の CALLF するための相対アドレスの計算だったのです．そして本来 INT 30H の割り込みベクタのあるべき場所には

```
EAH  E4H  05H  1DH  09H        JMPF  91D : 5E4H
```

（このアドレスはバージョンによって異なる）

という INT 21H ファンクションコールへのファージャンプ命令が入っています．INT 21H へのこのエントリはたとえば

```
POP   AX            ;PSP+05H の CALLF によるリターンオフセット
POP   AX            ;PSP+05H の CALLF によるリターンセグメント
POP   CS:WORK       ;CALL5 によるリターンオフセット
PUSHF               ;IRET 用リターンアドレスのセット
CLI                 ;
PUSH  AX            ;
PUSH  CX : WORK     ;
```

というようにスタック調整して

```
CMP    CL, 24H
JA     NORMAL_RET  …………何もせずすぐリターン
MOV    AH, CL
```

として，ファンクション 24H より大きなものははねてから本来の INT 21H のエントリへと入ってゆきます．したがって，COM 形式のプログラムのなかで CL にファンクション番号（0～24H まででなければならない）を入れて CALL 5 とすると，INT 21H ファンクションリクエストをしたのと同じ効果があるということです．しかしそのために MS-DOS システムは INT C0H と INT C1H の割り込みベクタを犠牲にしてしまいました．

MSDOS.SYS は，PSP を作るさいファンクション 26H，55H のルーチンを利用しています．

(4)　ファンクション 34H
――INT 21H 処理中フラグ（IN-DOS フラグ）の格納アドレスの取得――

呼び出し法

AH　　34H

リターン値

ES：BX　　INT 21H 処理中フラグ（IN-DOS フラグ）の格納アドレス

説明

INT 21H を実行すると，そのエントリでこの”INT 21H 処理中フラグ（IN-DOS フラグ）”をインクリメントしますので，フラグは 1 になります．したがって MS-DOS をマルチタスクで使用するように拡張するときにこのフラグを見て，0 であれば INT 21H をし，0 でなければ 0 になるまで待

つようにすれば，INT 21H を同時に多数のプログラムからコールすることが防げます．実際は外部コマンドの PRINT . COM でバックグラウンドでプリンタに出力するときに応用されています．

(5)　ファンクション 37H
――パラメータセパレータ，デバイス名許可フラグのリード/ライト――

呼び出し法

AH	37H	
AL	00H	パラメータセパレータの読み出し
	01H	パラメータセパレータの設定
	02H	デバイス名許可フラグの読み出し
	03H	デバイス名許可フラグの設定
DL	設定したいパラメータセパレータまたはデバイス名許可フラグ (AL＝01H，03H のとき)	

リターン値

DL	設定されているパラメータセパレータまたはデバイス名許可フラグ (AL＝00H，02H のとき)

説明

システムパラメータセパレータ（たとえば，DIR/W のように使うときの/のこと）の読み出しやセット，デバイス名(たとえば，CON，PRN 等)を許可するかどうかの設定・読み出しを行います．

AL←0 で実行すればセパレータの読み出しとなり，そのアスキーコードが DL にリターンされます．デフォールトの状態で実行すれば，/ (2FH) が読み出されます．AL←1 とすればセパレータの設定となり，逆に DL の中のアスキーコード文字がセパレータとして設定されます．

たとえば

```
MOV     DL, '%'
MOV     AL, 1      ;パラメータセパレータの設定
MOV     AH, 37H
INT     21H
```

とすれば，%がセパレータとして登録され，以降，たとえば DIR % W などとパラメータセパレータに%を使わなければならなくなります．ただし，システムコマンドはこのファンクションを使ってパラメータセパレータを読み出してチェックするようになっていますが，ユーザーやサードパーティの作成した外部コマンドはこのファンクションは当然ながら使っていない（未公開だから使っていないはず）ので，このファンクションによってはパラメータセパレータが変更されません．

デバイス名の許可フラグの設定・読み出しは，AL←2 で読み出し，AL←3 で設定となります．フラグの状態は DL レジスタを介してやりとりされ，0 なら不許可，0 以外なら許可を表わします．デフォールトではデバイス名許可となっており

```
A>DIR  >PRN
```

などとしたらプリンタへ出力結果が出ますが，デバイス名不許可とすると，ディスク上に PRN というファイル名でファイルを作成し，そのファイルへ出力するようになります．すなわち，ディスク名を特別扱いせずディスクのファイル名とみなすことになります．

ただし，ディスク名許可・不許可フラグの設定の機能は Ver 3. xx では使用できなくなっています．

(6)　ファンクション 50H，51H ——カレント PSP セグメントの設定・読み出し——

呼び出し法

AH　　50H，51H

BX　　カレント PSP に設定したい PSP の先頭セグメント（ファンクション 50H のとき）

リターン値

BX　　カレント PSP のセグメント（ファンクション 51H のとき）

説明

カレント PSP とは現在実行中のプログラムの PSP のことです．実行中のプログラムには，ファイルハンドル番号と内部 FCB 番号の対応表や INT 21H ファンクションコールをしたときの SS：SP の値などを格納するエリアがあるので，そのセグメントをシステム内に保存しておく必要があります．そのためのファンクションで 50H がカレント PSP アドレスの設定，51H が読み出しとなっています．カレント PSP アドレスの受け渡しは BX レジスタを介して行われます．

カレント PSP セグメントは MSDOS．SYS 内ではファンクション 26H または 55H の "PSP の作成" のときに更新され，プログラムの終了関係（ファンクション 0，31H，4CH，致命的エラーで Abort を選択したとき，^C の終了）を実行したとき 1 つ前の値に復旧されます．1 つ前のカレント PSP の値は現在のカレント PSP のオフセット 16，17H に格納されています．また，プログラムのロード&実行（ファンクション 4BH）でプログラム実行をしたときは，ファンクション 55H の "PSP の作成" がその中でコールされ，カレント PSP セグメントが更新されます．

(7)　ファンクションコール 52H ——システムパラメータ・テーブルアドレスの取得——

呼び出し法

AH　　52H

リターン値

ES：BX　　システムパラメータ・テーブルアドレス

説明

ここでいうシステムパラメータとは，① DPB 先頭アドレス，②内部 FCB 先頭アドレス，③

```
C>DEBUG
-A
53FB:0100 MOV AH,52
53FB:0102 INT 21
53FB:0104
-G=100,104          システムパラメータテーブル・アドレス

AX=5200  BX=00F0  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
DS=53FB  ES=091D  SS=53FB  CS=53FB  IP=0104   NV UP DI PL NZ NA PO NC
53FB:0104 BA31B8        MOV     DX,B831
-D91D:F0,FF
091D:00F0  F1 43 1D 09  07 00 1D 09-EA 00 60 00  B4 00 60 00   qC......j.'4
-Q         DPB先頭      内部FCB      CLOCKデバイ  デフォールトの
           アドレス     先頭アドレス スヘッダ・アド デバイスヘッダ・アドレス
C>                                 レス         (通常CONデバイスヘッダ)
```

図 3・3　システムパラメータテーブルの実際

FS：BX	内　　　容
−2, −1	最初の MCB パラグラフ・アドレス
0〜+3	最初の DPB アドレス
+4〜+7	最初の内部 FCB ブロック先頭アドレス
+8〜+B	CLOCK デバイスのデバイスヘッダ・アドレス
+C〜+F	IO. SYS 固有のデバイスヘッダ先頭アドレス
+10	接続ドライブ数
+11, 12	接続ドライブの内で最大のバイト/セクタ
+13〜+16	MSDOS. SYS 上限，ディスクバッファ先頭
+17〜+1A	最初のデバイスヘッダ（NUL）

図 3・4　システムパラメータテーブルの詳細

CLOCK デバイスアドレス，④デフォールトのデバイスヘッダアドレス，の 4 つです[注)]．1 つのアドレスはオフセット，セグメントの順に 4 バイトを占有し，計 16 バイトのテーブルが形成されます．

ファンクションコール 52H を実行すると，このテーブルのアドレスを ES：BX に得ることができます．**図 3・3，4** に実際にシステムパラメータ・テーブルを取得したところを示します．

(8)　ファンクションコール 53H——BPB → DPB 変換——

呼び出し法

AH　53H

DS：SI　BPB の先頭アドレス

ES：BP　変換された DPB を格納するバッファ先頭アドレス

リターン値

ES：BP　変換された DPB 先頭アドレス

説明

IO . SYS やデバイスドライバは BPB というデバイス（ディスク etc.）に関する諸元を格納したテーブルを持っています．BPB の内容については IO . SYS の章の**図 2・31** にあります．

MSDOS . SYS では BPB から DPB という形にして MSDOS . SYS のワークエリアに保存します．DPB の実際については**図 3・2** にあります．

BPB と DPB の違いは，BPB は予備のセクタや FAT 数，バイト/セクタ，セクタ/クラスタなど必要最小限のことしか格納されていませんが，DPB はこれらのデータに加えて，ディレクトリの総数やデータエリア先頭セクタ，$\log_2$（セクタ/クラスタ）の値まで BPB のデータから計算して DPB エリアの中に保存します．これらの値は処理の時点で必要になったときに計算するよりも，このようにあらかじめ計算しておけばディスク関係の処理が速くなる，という利点もあります．

さらに DPB にはカレントディレクトリを変更した場合のカレントディレクトリのパス名を格納するエリアもあります．

MSDOS . SYS が IO . SYS の BPB から DPB を作るときに用いるのがこのファンクションです．

注) この他にも**図 3・4** に示すようなシステム・パラメータがあります．

利用法はBPBが格納されているアドレスをDS：SIで指させ，変換されたDPBを格納したいアドレスをES：BPで指させてファンクション53Hを実行します．BPBはIO.SYSの"I/Oリクエストコール"で得られます．

具体的な変換法は以下のようになっています．**図3・2**と見くらべながら読んでください．

① ドライブ番号，論理装置コードの欄は飛ばす．

② バイト/セクタはそのまま転送

③ セクタ/クラスタは－1して転送

④ セクタ/クラスタのシフトカウント（$\log_2$（セクタ/クラスタ））を計算

⑤ 予備のセクタ数，FATの数はそのまま転送

⑥ データエリア先頭セクタは，まず

　CX←（バイト/セクタ）/32

　（32：1つのディレクトリエントリのバイト数）

　でディレクトリ数/セクタを算出して

　AX←（ディレクトリ数－1）＋CX

　CX←AX/CX

　でディレクトリのセクタ数を計算します．この値に

　AX←CX＋（セクタ/FAT）＊FATの数＋予備のセクタ数＋ディレクトリのセクタ数

　を加えて求めます．

⑦ データエリア総クラスタは

$$\frac{(\text{総セクタ/ディスク}) - (\text{データエリア先頭セクタ})}{(\text{セクタ/クラスタ})}$$

　で求めます．

⑧ セクタ/FATはそのまま転送

⑨ ディレクトリ先頭セクタは⑥のデータエリア先頭セクタの計算の途中で

　（ディレクトリのセクタ数）＋（セクタ/FAT）＊FAT数＋予備のセクタ数

　として求めます．

⑩ デバイスヘッダへのベクタは飛ばす．

⑪ メディアディスクリプタはそのまま転送

⑫ つぎのDPBへのポインタとカレントディレクトリパス名バッファは飛ばし，カレントディレクトリ変更中フラグを0クリアする

(9)　ファンクション58H ——MCB割り当て時フラグ——

呼び出し法

AH	58H	
AL	00H	フラグ読み出し
	01H	フラグ設定
BL	設定したいフラグ番号（AL＝1のとき）	

リターン値

BL	設定されているフラグ番号（AL＝0のとき）

フラグ番号の意味

0　できるだけ下位アドレスに MCB を割り当てる
1　もっとも小さいフリー MCB を割り当てる
2　できるだけ上位アドレスに MCB を割り当てる

説明

Ver 2. xx では未公開でしたが，Ver 3. xx では公開ファンクションコールとなりました．

MCB の割り当て要求が出た場合に，フラグの意味にしたがってメモリ割り当て法を設定するファンクションです．これを利用するとフリーメモリの途中にメモリを確保したりメモリブロックの指定の自由度が向上します．現在設定されている割り当て法番号を得たいときは，AL ← 0 にしてこのファンクションを実行すると，AX に現在設定されている値がリターンします．新しい値に設定したいときは設定したい値を BX に入れて，AL ← 1 として実行します．

(10)　ファンクション F8H
——OEM 拡張ファンクション処理アドレス設定——

拡張ファンクション処理用ルーチンを作り，そのエントリアドレスを DS：DX に入れてファンクション F8H を実行すると，それ以降ファンクション F9H～FFH を実行したときに，拡張用ファンクション処理ルーチンへ飛ぶようになります．拡張ルーチンでは AH にファンクション番号が入っているので，この番号によって処理を分けるようにします．リターンのしかたは IRET です．

拡張ルーチンを使えないようにしたいときは，DX ← FFFFH としてファンクション F8H を実行すれば，それ以降拡張ルーチンへ飛ばなくなります．

ただし，この機能は OEM 拡張用なので，将来メーカーがなんらかのサポートルーチンをこの機能の中に入れてくる可能性があります．

3・1・2　ディレクトリエントリと FCB

ディスク上にはデータやプログラムがどこから記録されているとか，その一連のプログラムやデータにつけた名前(ファイルネーム)などを記録する 1 項目 32 バイトの領域のテーブルがあります．この 1 項目を"ディレクトリエントリ"といい，ディレクトリエントリの集まりをディレクトリといいます．1 つのディレクトリエントリは**図 3・5** のようになっていて，ファイルネームのほかにもファイルの属性やそのファイルを作成変更した時刻や日付，ファイルサイズなども記録されています．

このディレクトリがディスク上のどこにあるかは，DPB を読み出すことによって知ることができます（前節の (2) の"DPB アドレスの取得"参照）．ディレクトリの先頭セクタの計算の仕方は

(予備のセクタ数)＋(FAT 数)＊(セクタ/FAT)　　セクタ
(＋6, 7)　　　　(＋8)　　　(＋0FH)

となります．下の () 内はディレクトリエントリ先頭からのオフセット値です．ディレクトリの最終セクタは

+0　　　+7+8　+A+B　+C　～　15+16，17+18，19+1A，　1B+1C～+1F

ファイル名	拡張子	属性	未使用システムリザーブ	最終変更時刻	最終変更日付	FAT 上の最初のクラスタNo.	ファイルサイズ

図 3・5　1 つのディレクトリエントリ

```
A>DEBUG
-A
116C:0100 MOV AH,1F   カレントドライブのDPBアドレスを求める
116C:0102 INT 21
116C:0104
-G,104       DPBアドレスはDS:BXに得られる

AX=1F00  [BX=43F1]  CX=0000  DX=0000  SP=FFEE  BP=0000  SI=0000  DI=0000
[DS=091D]  ES=116C  SS=116C  CS=116C  IP=0104   NV UP DI PL NZ NA PO NC
116C:0104 56            PUSH    SI
-D43F1                                  ディレクトリ・エントリ総数
091D:43F0     00 00 00 02 01 01 01-00 02 [70 00] 0C 00 7B 02    .........p...{.
091D:4400  02 [05 00] FC 00 60 00 04-00 4F 44 1D 09 00 00 00    ...|. ...OD.....
091D:4410  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00    ................
091D:4420  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00    ................
091D:4430  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00    ................
091D:4440  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 01    ................
091D:4450  01 00 02 01 01 01 00 02-70 00 0E 00 CA 02 03 07    ........p...J...
091D:4460  00 FC 00 60 00 05 FF FF-FF FF FF 00 00 E5 E5 E5    .|.`.........eee
091D:4470  E5          ディレクトリTOPセクタ                       e
-LES:0 0 5 7    7セクタ分                                   ディレクトリ先頭だ！
-DES:0          セクタ5から
116C:0000  49 4F 20 20 20 20 20 20-53 59 53 27 00 00 00 00    IO      SYS'....
116C:0010  00 00 00 00 00 00 5B 7E-31 07 02 00 00 A0 00 00    ......[~1.... ..
116C:0020  4D 53 44 4F 53 20 20 20-53 59 53 27 00 00 00 00    MSDOS   SYS'....
116C:0030  00 00 00 00 00 00 40 72-33 07 2A 00 00 45 00 00    ......@r3.*..E..
116C:0040  43 4F 4D 4D 41 4E 44 20-43 4F 4D 20 00 00 00 00    COMMAND COM ....
116C:0050  00 00 00 00 00 00 6E 82-33 07 3C 00 26 43 00 00    ......n.3.<.&C..
116C:0060  4B 4E 4A 44 49 43 20 20-53 59 53 20 00 00 00 00    KNJDIC  SYS ....
116C:0070  00 00 00 00 00 00 CA 72-34 07 4D 00 00 DC 00 00    ......Jr4.M..¥..
-DES:D90
116C:0D90  E5 E5 E5 E5 E5 E5 E5 E5-E5 E5 E5 E5 E5 E5 E5 E5    eeeeeeeeeeeeeeee
116C:0DA0  00 E5 E5 E5 E5 E5 E5 E5-E5 E5 E5 E5 E5 E5 E5 E5    .eeeeeeeeeeeeeee
116C:0DB0  E5 E5 E5 E5 E5 E5 E5 E5-E5 E5 E5 E5 E5 E5 E5 E5    eeeeeeeeeeeeeeee
116C:0DC0  00 E5 E5 E5 E5 E5 E5 E5-E5 E5 E5 E5 E5 E5 E5 E5    .eeeeeeeeeeeeeee
116C:0DD0  E5 E5 E5 E5 E5 E5 E5 E5-E5 E5 E5 E5 E5 E5 E5 E5    eeeeeeeeeeeeeeee
116C:0DE0  00 E5 E5 E5 E5 E5 E5 E5-E5 E5 E5 E5 E5 E5 E5 E5    .eeeeeeeeeeeeeee
116C:0DF0  E5 E5 E5 E5 E5 E5 E5 E5-E5 E5 E5 E5 E5 E5 E5 E5    eeeeeeeeeeeeeeee
116C:0E00  00 00 00 00 00 00 00 00-00 00 00 00 00 00 00 00    ................
-Q

A>
```

図3・6　ディレクトリエントリの実際

(データエリア先頭セクタ)－1

(+0B, 0CH)

ですからすぐ求めることができます．

実際にDEBUGによってディレクトリの位置をみつけ，ディレクトリの内容を表示させた例を**図3・6**に示します．

さて，ファイルをオープンするときにはFCB（File Control Block）というものを使用します．FCBは**図3・7**のようになっています．**図3・5**とくらべると内容が非常に似ていることがわかります．FCBはシステムがファイルをオープンするときにRAM上に作りますが，そのときはファイル名だけセットし，FCBの他のエリア（+0CH～24H）はゼロクリアしてファンクションコールします．するとMSDOS.SYSはディレクトリエントリから同じファイル名を持つものをさがし出してきて最終変更時刻などをコピーするため，このように似ているのです．それに加えて，FCBにはファイルのオープン状態（リードオンリかリードライトかなど）を示すビットと，いまそのファイルのどこを注目しているかを示すカレントブロックとカレントレコード，相対レコードというワークエリアがあります．

<table>
<tr><th>オフセット
(16進)</th><th colspan="2">内　　容</th></tr>
<tr><td>0</td><td colspan="2">ドライブ番号 (0…カレント, 1…A, 2…B, …)</td></tr>
<tr><td>1〜8</td><td colspan="2">ファイル名 (8文字)</td></tr>
<tr><td>9〜B</td><td colspan="2">拡張子 (3文字)</td></tr>
<tr><td>C, D</td><td colspan="2">カレントブロック</td></tr>
<tr><td>E, F</td><td colspan="2">レコードサイズ (一度に読み書きできるバイト数の設定)</td></tr>
<tr><td>10〜13</td><td colspan="2">ファイルサイズ</td></tr>
<tr><td>14, 15</td><td colspan="2">最終書込日付</td></tr>
<tr><td>16, 17</td><td colspan="2">最終書込時刻</td></tr>
<tr><td>18</td><td>bit7=0…ブロック型
ドライブ名</td><td>bit7=1…キャラクタ型
デバイス属性を8ビットに圧縮したもの</td></tr>
<tr><td>19〜1C</td><td>最初のクラスタ番号(19, 1AH)</td><td>対応するデバイスヘッダ・アドレス</td></tr>
<tr><td>1D</td><td rowspan="3">} 最初のクラスタ番号
} カレントディレクトリのクラスタ</td><td rowspan="3"></td></tr>
<tr><td>1E</td></tr>
<tr><td>1F</td></tr>
<tr><td>20</td><td colspan="2">カレントレコード</td></tr>
<tr><td>21〜24</td><td colspan="2">相対レコード</td></tr>
</table>

拡張FCBは上記FCBの先頭に下の7バイトのエリアをつけたもの.
拡張FCBはシステムファイルやボリュームラベルへアクセスするとき使う.

オフセット (16進)	内　　容
0	FFH 拡張FCBを表わすマーク
1〜5	システムリザーブ (現在未使用)
6	属　性

図3・7　FCBの内容

3・1・3　PSPの構造と内部FCB⟷ファイルハンドル対応表

MSDOS.SYSと外部コードの間をとりもつためのシステムワークエリアとして, PSP (Program Segment Prefix) と呼ばれる256バイトのエリアがあります.

PSPの中には**図3・8**のような内容が記録されていますが, この中で一般にはあまり知られていなくて重要なものに"ファイルハンドル⟷内部FCB対応表"があります. ファイルをアクセスするときにはFCBを使う方法がありますが, もう1つ"ファイルハンドル"を使う方法があります.

ファイルハンドルによると, 1度ファイル名を指定してオープンしたあとは, そのファイル名に対応する"番号"を指定することによって, データのリード/ライトなどファイルに関する操作ができるようになります. この"番号"のことをファイルハンドルといいます. ファイルハンドルを使うとFCBは必要ないように思えますが, 実はMSDOS.SYSの内部では"内部FCB"を作ってFCBによ

<table>
<tr><th>オフセット
(16 進)</th><th colspan="2">内　　　容</th></tr>
<tr><td>0, 1</td><td colspan="2">CD 20…INT 20H のことで COM 型終了命令
すなわち，COM 型なら JMP 0 で終了できる</td></tr>
<tr><td>2, 3</td><td colspan="2">利用可能最大セグメント＋1（RAM上限）</td></tr>
<tr><td>4</td><td colspan="2">0</td></tr>
<tr><td>5〜9</td><td colspan="2">MSDOS . SYS 内部 INT 21H のファーコール用エントリへのファーコール命令
実際は JMPF 0：C0 に相当する命令が入っていて 0：C0〜C5 に JMPF MSDOS. SYS のセグメント：INT 21H ファーコールエントリに相当する命令が入っている．このため INT 30H と INT 31H がつぶされている．COM 型プログラムで CALL 5 のニアコールで INT 21H ファンクションコールができる</td></tr>
<tr><td>A〜D</td><td colspan="2">プログラム終了アドレス（INT 22H）のコピー
COM 型で JMP 0 をしたり INT 20H したりファンクション 0，ファンクション 4CH などのプログラムの終了をすると，このプログラムの終了アドレスへファージャンプする</td></tr>
<tr><td>E〜11</td><td colspan="2">プログラム中断アドレス（INT 23H）のコピー
プログラム中断アドレスとは [CTRL] ＋ [C]（または [STOP] キー）を押したときに INT 23H されるアドレスのこと</td></tr>
<tr><td>12〜15</td><td colspan="2">致命的エラー処理アドレス（INT 24H）のコピー
ディスクのエラーなどのとき自動的にコールされる．通常 COMMAND . COM 内にあって（A）abort（R）retry（I）ignore と聞いてくるルーチンのアドレス</td></tr>
<tr><td>16, 17</td><td colspan="2">この PSP を作ったプログラムの PSP のセグメント．もし COMMAND . COM から起動されていれば COMMAND. COM 常駐部のセグメントとなる．</td></tr>
<tr><td>18〜2B</td><td colspan="2">ファイルハンドラ ↔ 内部 FCB 対応表．20 バイトある．すなわち，ファイルハンドルを使って一度にオープンできるファイルは 20 個までというのはここに原因がある</td></tr>
<tr><td>2C, 2D</td><td colspan="2">環境エリアのセグメント</td></tr>
<tr><td>2E〜31</td><td colspan="2">システムワークで INT 21H したときの SS：SP の退避に使っている</td></tr>
<tr><td>32〜4F</td><td colspan="2">システムワークだが未使用のようである</td></tr>
<tr><td>50〜52</td><td colspan="2">CD 21 CB…INT 21H RETF
プログラムから CALLF（PSP のセグメント）：50H とファーコールによって INT 21H ファンクションコールができるようにしてある</td></tr>
<tr><td>53〜5B</td><td colspan="2">システムワークだが未使用のようである</td></tr>
<tr><td>5C〜6B</td><td>デフォルトの FCB1</td><td rowspan="2">FCB としては与えられたバイト数が少ないので，FCB として使うにはどちらか一方の FCB を没にしなければならない．外部コマンドへのファイル名の引数渡しエリアと考えたほうがよい</td></tr>
<tr><td>6C〜7F</td><td>デフォルトの FCB2</td></tr>
<tr><td>80〜FF</td><td colspan="2">デフォルトの DTA
プログラム起動時はコマンドラインの文字列が入る
80H が文字数（最後の 0DH は含まない）
81〜FFH コマンドライン文字列</td></tr>
</table>

図 3・8　PSP の内容

ってファイルをアクセスしているのです.

内部 FCB には先頭から番号がつけられており，ファイルハンドル番号がどの内部 FCB に対応しているかを表わしています. すなわち, PSP の 18H にはファイルハンドル 0 に対する内部 FCB 番号が入っていて，19H にはファイルハンドル 1 に対する内部 FCB 番号というように 2BH まで続きます. 通常 MS-DOS のイニシャライズの時点で内部 FCB の 0 番には AUX, 1 番には CON, 2 番には PRN がすでにオープンされていますので，標準入出力，エラー出力，プリンタ出力などに対応して

18H （ファイルハンドル0）………1（内部 FCB1 番 CON）標準入力
19H （ファイルハンドル1）………1（内部 FCB1 番 CON）標準出力
1AH （ファイルハンドル2）………1（内部 FCB1 番 CON）標準エラー出力
1BH （ファイルハンドル3）………0（内部 FCB0 番 AUX）標準補助装置
1CH （ファイルハンドル4）………2（内部 FCB2 番 PRN）標準プリンタ

の値がセットされています. オープンされているファイルが対応していないファイルハンドルには FFH が入っています. このようにファイルハンドルと内部 FCB の対応表が PSP の 18H～2BH の 20 バイト分しかないために，ファイルハンドルで同時にオープンできるファイルの数は 20 個までなのです.

3・1・4 内部 FCB

ファイルハンドルを使うと FCB は使わなくてすみますが, MSDOS . SYS の内部では"内部 FCB" というものをシステムワークエリアの中に作ってファイルをアクセスしています. **図 3・7** に FCB を示しましたが，内部 FCB ブロックは内部 FCB の集まりにリンク情報をつけた形式をしており，**図 3・9** のようになっています. **図 3・9** で 6～2DH が 1 つの内部 FCB で, ブロックに 2 つ以上内部 FCB があれば 2EH 以降 6～2DH と同じ内容が繰り返されます. 内部 FCB ブロックのリンクは，あるまとまった数の内部 FCB を単位として行われます. 内部 FCB1 個について見ると，通常の FCB の前に 3 バイトのシステムリザーブエリアが付加されますが，これは Ver 2. xx で Ver 3. xx になると 17 バイトになっています.

このシステムリザーブエリアは

① いくつのファイルハンドルから使用されているか
② リード・ライトの許可フラグ

オフセット (16 進)	内　　容
0～3	つぎの内部 FCB ブロックへのアドレス (最終ブロック時 (0, 1)＝FFFF)
4, 5	このブロック内での内部 FCB の数
6	いくつのファイルハンドルから使用されているか.
7	0…リードオンリ，1…ライトオンリ，2…リード・ライト
8	属　性
9～2D	通常の FCB とおなじ（図 3・7 参照）

（0～2D：内部 FCB ブロック／6～2D：1 つの内部 FCB，内部 FCB の数 (4, 5) だけくり返される）

図 3・9 内部 FCB ブロック

③　ディレクトリ属性

からなっています．①については，1つの内部 FCB がいくつかのファイルハンドルから参照されていることがあり，ファイルハンドルでオープンをするときこの項がインクリメントされ，クローズするときデクリメントされ，0になるまで内部 FCB はクローズされません．くわしくはオープン・クローズの解析のほうを参照してください．

内部 FCB は先頭から順番に番号がつけられ，システムイニシャライズ時にすでに AUX，CON，PRN の順番で内部 FCB はオープンされています．

内部 FCB の先頭ブロックのアドレスは，未公開 INT 21H ファンクション 52H で得ることができます．

```
MOV   AH, 52H
INT   21H
```

を実行すると，ES：BX にシステムパラメータテーブルのアドレスを返してくるので，内部 FCB 先頭ブロックアドレスはその＋4～＋7にオフセットアドレス，セグメントアドレスの順に格納されていますから

```
LDS   SI, dword    ptr   ES：4 [BX]
```

で DS：SI レジスタに取り込むことができます．これを利用して内部 FCB のファイル名部分を出力するプログラムを**図 3・10** に示しておきますので，内部 FCB の構造を理解する上での参考にしてください．**図 3・11** はその実行結果です．

3・1・5　ディスクバッファ

ディスクバッファの数は CONFIG．SYS の BUFFERS で指定できます．指定しなければ2個確保されます．指定は 99 個まで可能です．1つのバッファは接続されているディスクのセクタ長の中で最大長＋16 バイトのメモリを必要としていますから，99 個もとればバッファだけで5インチ 2DD なら1セクタ 512 バイトなので

$(512+16)*99=52272$

約 53k バイトも消費してしまいます．また，5インチ 2DD ではディスク交換の判別ができませんので，バッファリングの効果は期待できません．

バッファは**図 3・12** のようにヘッダ 16 バイトを持っています．このヘッダ部分にはつぎのバッファへのリンクアドレスや，どのドライブのバッファか，まだディスクへ書き込まずにバッファリン

ディスク交換の判定

”I/O リクエストコール”のコマンドの中に「ディスク交換の判定」というものがありますが，5インチ 2DD ではディスクが交換されていないという判定ができません．

交換すれば「交換された」という情報を返しますが，交換しなくても「不明」という情報しか返しません．これは節電のために一定時間後にモータの回転を止めてしまうことと深い関係があります．ディスク交換はハード的にインデックスホールの通過によって判定しています．すなわち，インデックスホールがつねに一定時間ごとに検出されていれば，ディスクは取り出されずにディスクユニットの中に挿入されたままだということがわかります．ところが5インチ 2DD ユニットではモータの回転を止めてしまうので，交換されたためにインデックスホールが検出されなくなったのか，節電のためにモータの回転を止めたのかわからなくなり，「交換不明」ということになるのです．

```
;
; Display inner FCB
;       for MS-DOS version 2.XX

code segment

assume cs:code,ds:code

        org 100h

lf      = 10
cr      = 13

print   Macro S             ; Print strings
push    ds
push    cs
pop     ds
mov     dx,offset S
mov     ah,9
int     21h
pop     ds
Endm

space   Macro               ; Print space
mov     dl,' '
mov     ah,2
int     21h
Endm

INFCB   proc near

mov     ah,52h              ; Get system parameter table address
int     21h
lds     si,es:4[bx]         ; In-fcb's link top address
InFcbLoop10:
mov     bx,ds
mov     ax,si
mov     es,bx               ; save link address
mov     di,ax
call    PrintDoubleWord ; Print In-fcb' block link address

call    crlf
mov     ax,4[si]            ; In-fcb Number of this block
call    PrintWord
print   message1            ; FCB's exist in this block
mov     cx,4[si]
add     si,6
InFcbLoop15:
push    cx
mov     al,[si]             ; How many file-handle use this In-fcb
call    PrintByte
space
mov     al,1[si]            ; 0 read-only , 1 write-only , 2 read-write
call    PrintByte
space
mov     al,2[si]            ; attribute
call    PrintByte
space
mov     al,3[si]            ; Drive number
call    PrintByte
space
mov     cx,11
```

図 3・10　内部 FCB のファイル名出力プログラム (1/2)

```
mov     bx,4            ; File name point
InFcbLoop20:
mov     dl,[si][bx]
inc     bx
mov     ah,2
int     21h
loop    InFcbLoop20
call    crlf
add     si,28h
pop     cx
loop    InFcbLoop15

lds     si,es:[di]      ; Next In-fcb address link
cmp     si,-1           ; Link end ?
jne     InFcbLoop10

mov     ax,4c00h        ; Program end.
int     21h

INFCB   ENDP

message1 db " FCB's exist in this block",cr,lf,"$"

crlf    proc near       ; Print CR LF subroutine
push    ds
push    cs
pop     ds
mov     dx,offset crlfstring
mov     ah,9
int     21h
pop     ds
ret
crlfstring db cr,lf,"$"
crlf    endp

include binhex.asm

code    ends

        end INFCB
```

図 3・10 内部 FCB のファイル名出力プログラム (2/2)

```
C>infcb
091D:0007
0005 FCB's exist in this block
03 02 00 00 AUX
09 02 00 00 CON
03 01 00 00 PRN
00 00 20 03 INFCB    COM
00 02 00 03 DPB2     COM
0F7E:0000
000F FCB's exist in this block
00 00 20 03 BINHEX   ASM
00 00 00 00
00 00 00 00
00 00 00 00
00 00 00 00
00 00 00 00
00 00 00 00
00 00 00 00
00 00 00 00
00 00 00 00
00 00 00 00
00 00 00 00
00 00 00 00
00 00 00 00
00 00 00 00
```

図 3・11 内部 FCB の実例（図 3・10 プログラムの出力例）

オフセット (16進)	内　　　容	
0～3	つぎのバッファへのリンクアドレス	
4	このバッファを使用しているドライブ番号 (FFHのときは未使用)	
5	このバッファにディスクへ書き込むべきデータを持っていることを表わすフラグ (書き込むべきバッファのとき1)	
6	〃	
7	汎用処理フラグ	
8, 9	このバッファに読み込んでいるセクタ番号	
A	FATの数	バッファにFATやディレクトリを読み込んだときに使用される.
B	セクタ/FAT	
C～F	このバッファを使用しているディスクのDPB先頭アドレス	
10～20F	バッファ本体. マシンに接続されているディスクの中でもっとも大きいセクタのバイト数に合わされる.	

図3・12　ディスクバッファの内容

グ中である, などのフラグがあります.

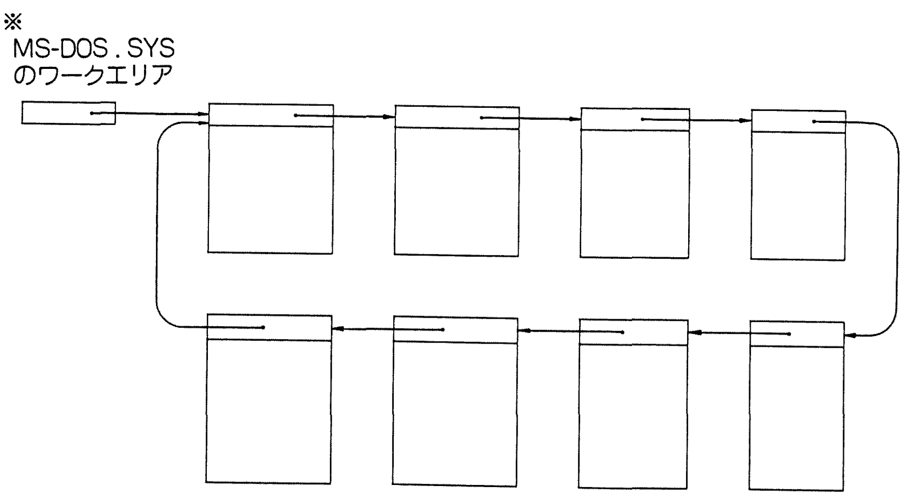
ここで簡単にディスクバッファの内容について触れておきましょう.

① つぎのバッファへのリンクアドレス (+0～+3)

バッファは最大99個とれますが, 各バッファはこのリンク情報によって**図3・13**のようにリング状につながれています.

Ver 2.xxならバッファ先頭を指すポインタはMSDOS.SYS内にあって, "システムパラメータテーブル"のアドレス+13H～16Hに格納されています.

```
MOV   AH, 52H
INT   21H
```



※アドレスはシステムパラメータ・テーブル+13H～16Hで求まる.

図3・13　ディスクバッファのリンク概念図 (Ver 2.XX)

LDS　SI, ES : [BX+13H]

で DS : SI にバッファ先頭を指すポインタ格納アドレスが求まります．

これを利用してディスクバッファのヘッダと内容を表示するプログラムを**図 3・14** に，実行結果を**図 3・15** に示します．このポインタはバッファをアクセスするごとにつぎのバッファを指すように移動します．このプログラムを実行してみると，ディスクをアクセスするコマンドを実行するたびにバッファが少しずつずれていくのがわかるでしょう．たとえば DIR/W ならディレクトリと FAT をアクセスするので，バッファが 3 つずれていきます．

```
;
;  Display disk buffer
;
; Usage : A>DBUF       Display disk buffer link
;         A>DBUF X     Display disk buffer contents
;

code    segment
assume  cs:code,ds:code

        org 100h

DBUF    proc

mov     si,80h
mov     di,offset flag
movsb
mov     ah,30h
int     21h
mov     si,13h
cmp     al,3
jne     DBUF10
dec     si      ; si<=12h
DBUF10:
mov     ah,52h
int     21h
lds     si,es:[bx][si]
DBUFLoop:
call    dump
cmp     word ptr [si],-1
je      DBUF_END
lds     si,[si]
jmp     DBUFLoop
DBUF_END:
mov     ax,4c00h
int     21h

flag    db 0
DBUF    endp

dump    proc

push    si
lodsw
mov     bx,ax
lodsw
xchg    ax,bx
call    PrintDoubleWord
call    spcl
call    spcl
```

図 3・14　ディスクバッファのヘッダ内容表示プログラム（1/2）

```
call    spcl
mov     cx,8
DumpLoop1:
push    cx
lodsb
call    PrintByte
call    spcl
pop     cx
loop    DumpLoop1
lodsw
mov     bx,ax
lodsw
xchg    ax,bx
call    PrintDoubleWord
call    crlf

cmp     cs:flag,0
je      DumpEnd
cmp     cs:flag,1
je      DumpEnd
mov     cx,16
DumpLoop2:
push    cx
mov     cx,16
push    si
DumpLoop3:
push    cx
lodsb
call    PrintByte
call    spcl
pop     cx
loop    DumpLoop3
call    spcl
pop     si
mov     cx,16
DumpLoop4:
push    cx
lodsb
cmp     al,20h
jae     Dump10
mov     al,'.'
jmp     Dump20
Dump10:
cmp     al,80h
jb      Dump20
mov     al,'.'
Dump20:
mov     dl,al
call    OutChar
pop     cx
loop    DumpLoop4

call    crlf
pop     cx
loop    DumpLoop2
call    crlf
DumpEnd:
pop     si
ret

dump    endp

spcl    proc
mov     dl,' '
call    OutChar
ret
spcl    endp

OutChar proc
mov     ah,2
int     21h
ret
OutChar endp

crlf    proc
mov     dl,13
call    OutChar
mov     dl,10
call    OutChar
ret
crlf    endp

include binhex.asm

code    ends

end     DBUF
```

図 3・14 ディスクバッファのヘッダ内容表示プログラム (2/2)

② このバッファを使用しているドライブ番号（+4）とこのバッファに読み込んでいるセクタ番号（+8, 9）

ディスクが交換されていないことがI/Oリクエストコールによってわかると，この項目を全バッファについてサーチして，読み込みたいドライブとセクタ番号と同じものがあれば，新たにディスクから読み込むようなことはせずに，そのバッファ内の内容をそのまま利用します．

つぎに述べる(+5)書き込みフラグが1のときは，書き込むドライブとセクタ番号を表わします．

ドライブ番号がFFHのときは，このバッファは未使用を表わします．

③ このバッファにディスクへ書き込むべきデータを持っていることを表わすフラグ（+5）

ディスクへの書き込みが発生した場合，すぐには書き込まずにバッファにためます．そのようなバッファは未フラッシュのデータを持っているということを表わすために，この項(+5)が1にされます（ドライブ番号，セクタ番号は（+4）と（+8, 9）に書かれる）．

```
C>dbuf
11B4:0000    FF 00 00 01 07 00 01 00 0D69:0C88
1172:0000    FF 00 00 01 01 00 02 03 0D69:0C88
1049:0000    FF 00 00 01 CB 01 01 00 0D69:0C88
0FE6:0000    02 00 00 01 CB 01 01 00 0D69:0C88
1193:0000    02 00 E5 01 09 00 01 00 0D69:0C88
1007:0000    02 00 00 01 0A 00 01 00 0D69:0C88
11D5:0000    02 00 E5 01 0B 00 01 00 0D69:0C88
1028:0000    02 00 00 01 0C 00 01 00 0D69:0C88
108B:0000    02 00 00 01 0D 00 01 00 0D69:0C88
106A:0000    02 00 00 01 CA 00 01 00 0D69:0C88
11F6:0000    02 00 00 01 78 00 01 00 0D69:0C88
0FC5:0000    02 00 00 01 7A 00 01 00 0D69:0C88
110F:0000    02 00 E5 01 7C 00 01 00 0D69:0C88
10CD:0000    02 00 00 01 7E 00 01 00 0D69:0C88
091D:41E1    02 00 00 01 01 00 02 03 0D69:0C88
10AC:0000    02 00 00 01 82 00 01 00 0D69:0C88
1151:0000    02 00 00 01 84 00 01 00 0D69:0C88
0FA4:0000    02 00 00 01 07 00 01 00 0D69:0C88
1130:0000    02 00 E5 01 08 00 01 00 0D69:0C88
FFFF:FFFF    02 00 00 01 AE 00 01 00 0D69:0C88
```

図 3・15 図3・14の実行結果（ディスクバッファヘッダの内容）

処理がひと区切りつくと全バッファがサーチされ，この項が1になっているバッファがディスクに書き込まれ，この項は0にされます．

④ FATの数（+0AH），セクタ/FAT（+0BH）

ディレクトリやFATをとくに高速に扱うために，FATの数とセクタ/FAT情報をDPBからここへ複写しておくのです．

⑤ このバッファを使用しているディスクのDPBアドレス（+0C～0FH）

ディスクをアクセスするさいに必要となるディスクの諸元を格納したDPBをすぐにサーチできるように，DPB先頭アドレスをここに複写しておくのです．

⑥ バッファ本体（+10～20FH）

ディスクとやりとりするデータを実際に格納しておくバッファ本体です．立ち上がりにおけるイニシャライズ時に，マシンに接続されているディスクユニットの中で最大のバイト/セクタのものに合わされます．5インチ2DDでは512バイト/セクタ，8インチ2D(5インチ2HD)で1024バイト/セクタです．したがって，1台でも8インチ2D (5インチ2HD) が接続されていれば，バッファ本体のアドレスは+10～40FHとなります．

3・1・6 NULデバイス

CON, AUX, PRN, CLOCKの各デバイスのデバイスヘッダと処理ルーチンは，IO.SYS内にありますが，NULデバイスだけはMSDOS.SYS内にデバイスヘッダと処理ルーチンを持っています．

● NULデバイスのデバイスヘッダ

3・1・1(7)のファンクション52Hで示したシステムパラメータの+17～27Hにあります．NULデバイスのデバイスヘッダの先頭をオフセット0とすると

+0～+3　　つぎのデバイスヘッダアドレス

```
+4, 5        デバイス属性 8004H
                       ↑  ↑
                       │  └──NUL デバイス
                       └─────キャラクタ型
+6, 7        ストラテジエントリ
+8, 9        割り込みエントリ
+0A, 11H     NUL　デバイス名
```

となっています.

● **NUL ストラテジエントリ**

単にシステムのワークエリアにコマンドパケットアドレス ES：BX の値を保存するだけです．すなわち

```
MOV   CS : WORK, BX
MOV   CS : WORK+2, ES
RETF
```

というような処理をしています.

● **NUL 割り込みエントリ**

単にコマンドパケットのステータスにエラーなし，DONE ビットを立てるだけです．すなわち

```
PUSH    ES
PUSH    BX
LES     BX, CS : WORK
OR      WORD  PTR  3 [BX], 100H
                           ↑
                        DONE ビット
POP     BX
POP     ES
RETF
```

というような処理をしています.

結局 NUL デバイスはなにもしない割り込みルーチンというだけであって，NUL デバイスも MS-DOS のデバイスのルールに従っているのです．MSDOS . SYS 内で特別にデバイス名 NUL を判断してデータを捨てているわけではありません.

3•2　INT 21H ファンクションコールの解析

3・2・1　INT 21H エントリ

INT 21H の割り込みベクタはセグメント 0，オフセット 84H～87H にあり，MSDOS . SYS の INT 21H 処理エントリを指しています．INT 21H 処理のエントリ部分のフローチャートを**図 3•16** に示します.

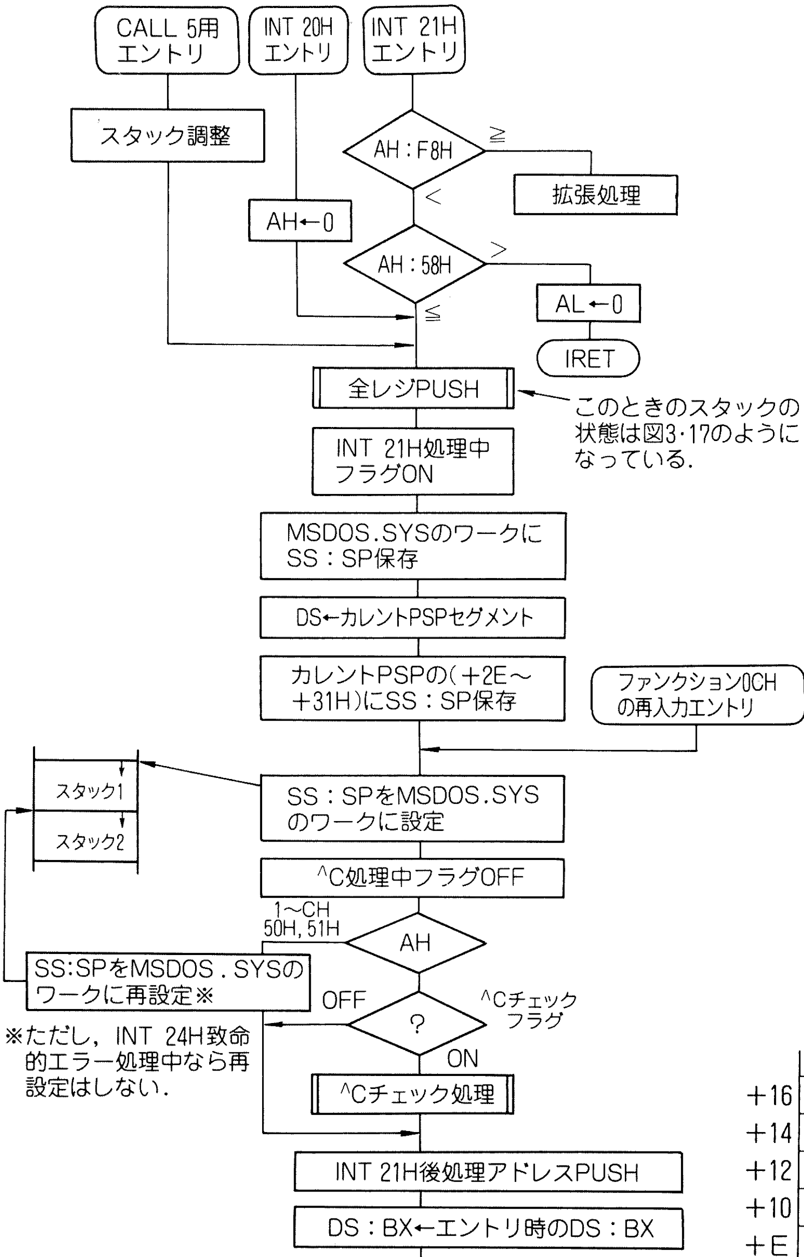


図 3・16　INT 21H ファンクションコールエントリ部分後処理

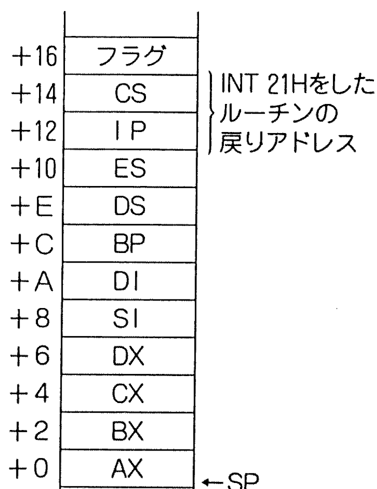


図 3・17　INT 21H 各処理ルーチン
エントリ時のスタック状態

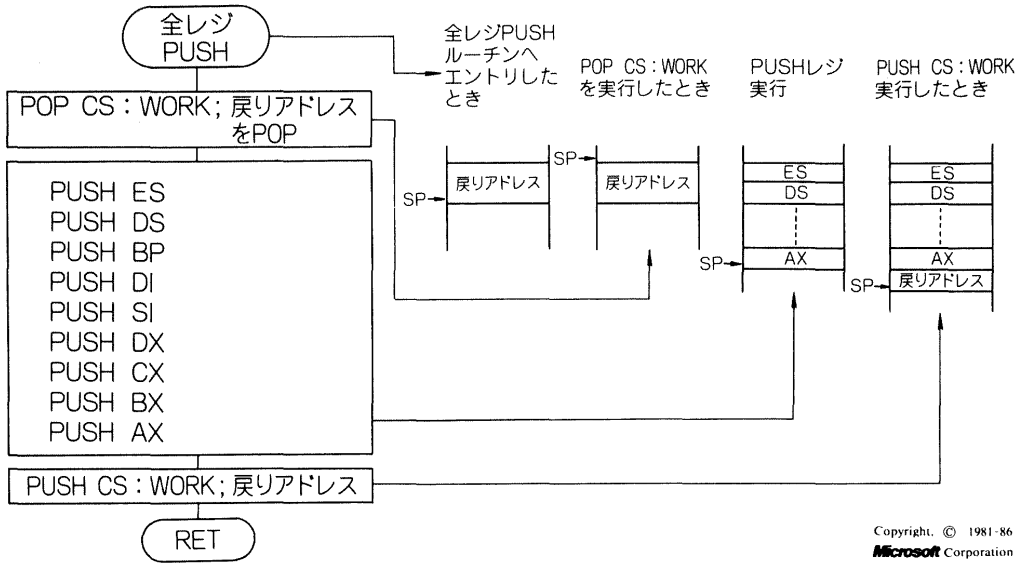


図 3・18 全レジスタ PUSH ルーチン

各処理ルーチンにエントリするときは，スタックには**図 3・17** のようにエントリ時のレジスタの値が格納されており，各処理ルーチンでエントリ時のレジスタの値が必要になったときには，スタック上の値を参照します．逆にスタック上の対応するレジスタのところに書き込んでおけば，リターンするときに対応するレジスタに値が POP されてもどるようになっています．

INT コールではフラグをリターンすることはできませんが，対応するスタック上のフラグのビットをアクセスすれば，リターン値としてキャリーフラグなどを返すこともできます．また，処理ルーチンにエントリするときの DS : BX の値が入っています．

また，カレント PSP の(+2E～31H)に全レジスタ PUSH したあとの SS : SP を保存しているのもわかります．

ファンクション 01～0CH と 50, 51H で SS : SP を再設定しているのは，ファンクション 0CH がキーバッファを空にしたあとリエントラントに INT21H のファンクション(1, 6, 7, 8, 0AH)を実行するためです．

全レジスタ PUSH ルーチンで PUSH されるレジスタは AX, BX, CX, DX, SI, DI, BP, DS, ES です．このルーチンは**図 3・18** のようになっており，CALL で呼び出すことのできるサブルーチンとなっています．**図 3・19** に，このルーチンで PUSH したレジスタを復帰する全レジスタ POP ルーチンを示します．全レジスタ POP は INT 21H の後処理ルーチンで使われます．

INT 21H 後処理ルーチンのアドレスを PUSH して各処理ルーチンへジャンプしますので，各処理ルーチンの処理が終わって RET が実行されると，INT 21H 後処理ルーチンへリターンすることになります．

ファンクション番号→各処理ルーチンのアドレス変換はテーブルを用いて行っています．

図 3・20 に，INT 21H 後処理ルーチンのフローチャートを示します．はじめに割り込みフラグをクリアしてスタックをエントリ時の値に復帰しますが，最後の IRET 命令で INT 21H を実行したルーチンへリターンすると同時に，そのときのフラグの状態が復帰されることになります．

INT 21H 各処理ルーチンのためのスタックは MSDOS . SYS ワークエリア内にとりますので，1

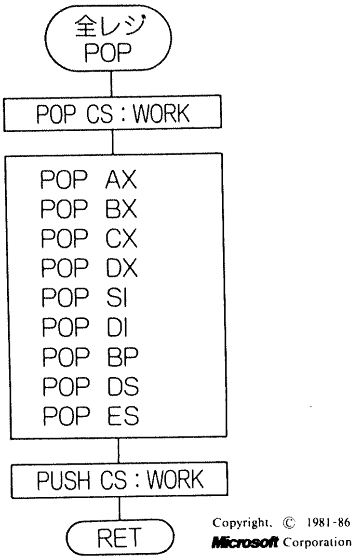


図 3・19　全レジスタ POP ルーチン

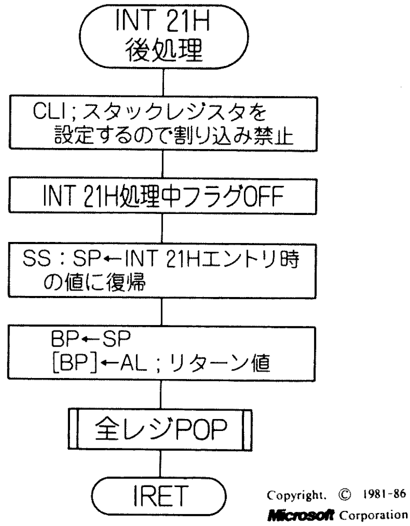


図 3・20　INT 21H 後処理フローチャート

回の INT 21H コールで必要なユーザースタックは**図 3・17** に示される 18H(=24)バイトということになります．

3・2・2　CALL 5 用エントリ

COM 型プログラムは，CALL 5 によって INT 21H ファンクションの 0〜23H までを呼び出すことができます．この場合，ファンクション番号は AH ではなく CL に入れます．これは往年の 8 ビット OS　CP/M との互換性を考慮して作られたものです．

さて，CALL 5 というのは，PSP のオフセット 5 に書いてある命令を呼び出すことに相当しますが，PSP のオフセット 5 には，たとえば

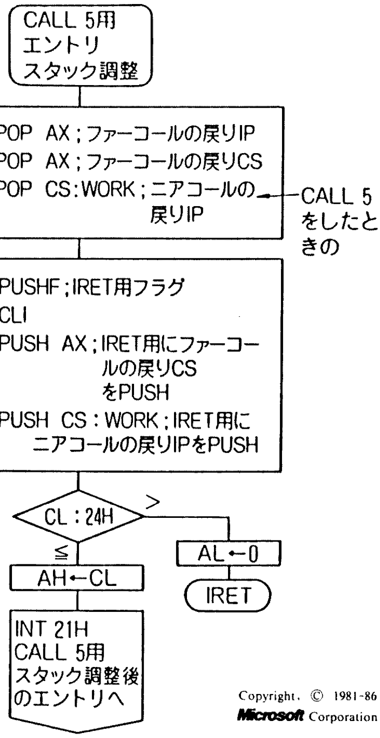
```
1580 : 0005    9AF0FE1DF0   CALL   F01D : FEF0
```

というような FAR コール命令が入っています．これは飛び先がわからないようにセグメントとオフセットの組合せをめちゃくちゃな値にしているのですが

```
    F01D
 +) FEF0
 -------
  1000C0
```

という計算をすれば，物理アドレス 0000 : 00C0H を指していることがわかります．

0000 : 00C0H は本来 INT 30H の割り込みベクタの格納アドレスですが，MSDOS . SYS のイニシャライズ時に 0000 : 00C0H 番地には，INT 21H の CALL 5 用エントリへのファージャンプが格納されています．ファージャンプは 5 バイトを必要としますので，INT 30H と INT 31H がつぶされてしまいます．INT 21H の CALL 5 用エントリは**図 3・16** に示されています．ただし，**図 3・16** に示されるように，INT 21H の IRET でリターンするよう**図 3・21** のようなスタックの調整が行われます．ニアコールはもどりアドレスのオフセット，ファーコールはもどりアドレスのオフセットとセグメントしかスタックに待避しないのに対して，INT コールはもどりアドレスのオフセットとセグメントに加えて，フラグもスタックに待避します．したがってニアコールのオフセットとファーコールのセグメントを合わせて IRET のもどりアドレスとしています．おまけに INT コールしたと



注）WORKは2バイトのワーク
WORK DW ?

図 **3・21** **CALL 5** 用エントリ
スタック調整フローチャート

きに自動的に割り込みフラグをクリアしますので，CALL 5用エントリの調整ルーチンの中でCLIが実行されます．

CALL 5でもファンクション0～23Hを実行することができますが，8ビットマシン用のOS CP/Mのシステムコールをエミュレートする化石のようなファンクションコール機能であり，MS-DOS特有のパス名とファイルハンドルによるファイルアクセスなどを使えないので，これからプログラムを作るにはCALL 5よりもINT 21Hを使うべきです．

前に述べたように，PSPのオフセット5にあるファーコールアドレスはセグメントとオフセットの組合せが一見してわからないようにカモフラージュされていますが，MSDOS．SYSのPSP作成ルーチンのこのカモフラージュアドレス計算法にバグがあって，0000：00C0Hへのファーコールではなく0000：00BEHへのファーコール命令がPSPのオフセット5に書き込まれることがあります．そうすると，通常INT 2FHに対する割り込みベクタは0000：0000Hがセットされていますので

```
0000:00BE     0000          ADD  [BX+SI], AL
0000:00C0     EAXXXXXXXX    JMP  XXXX:XXXX
                                 CALL 5用エントリアドレス
```

という命令が実行され，ファンクションコールは実行されるけれども

```
ADD    [BX+SI], AL
```

というよけいな命令を実行してしまい，DS：[BX+SI]の示すメモリをこわしてしまうということが起こってしまいます．

したがって，CP/Mから移植してきてどうしてもCALL 5を使う必要がある場合は，PSPのオフセット5に書いてある命令が0000：00C0Hへのファーコールになっていることを確かめて使ったほうがよいでしょう．逆にPSPのオフセット5のファーコールのアドレスを直接0000：00C0Hに変えてもよいでしょう．

ファンクション00H

プログラムの終了
関連ファンクション31H, 4CH, 致命的なエラーのAbort

プログラムの終了には，終了時にプログラムの占有していたメモリを解放するものとしないものがあります．メモリを解放しないものを常駐終了といいます．

プログラムの終了に関するファンクションにはつぎの3つがあります．

① ファンクション0　　プログラムの終了
② ファンクション31H　常駐終了
③ ファンクション4CH　プロセス終了

これら3つのエントリは異なりますが，フラグを立てて3つの処理を区別するのと，ちょっとした前処理ののち，共通の終了処理ルーチンへと導かれます．またこのフラグは子プロセスの終了コードの上位1バイトになります．またファンクションではありませんが，致命的エラーのAbortが実行されても終了処理の共通部分へエントリします．

共通処理では，PSP内に保存していたINT 22H～24Hのベクタをセグメント0のINT 22H～24Hのベクタアドレスに復旧することや，ファンクション0と4CHではプログラム領域のメモリの解放や20個の全ファイルハンドルのクローズなどが行われます．このとき，PSPにファンクション0ではINT 21Hエントリ時にスタックに待避されたリターンCS(**図3・17**参照)を，ファンクション31Hと4CHではカレントPSPのセグメント(3・1・1の(6)参照)で示されるPSP(オフセット0～FFH)を使用するのです．したがってファンクション0では，PSPがCSセグメントの0～FFHにあるCOM型でなければうまく終了できないのです．そして最後にCPUのフラグを

```
1111 0010 0000 0010
XXXX ↑↑↑↑ ↑↑↑↑ ↑↑↑↑
                  └── CF キャリー
                 └─── X(このビットはいつでも1)
                └──── PF パリティ
               └───── X
            └──────── AF 補助キャリー
           └───────── X
          └────────── ZF ゼロ
         └─────────── SF サイン
       └───────────── TF トラップ
      └────────────── F インタラプト
     └─────────────── DF ディレクション(方向)
    └──────────────── OF オーバフロー
```

(X：Don't Care)

ファンクション番号	0	31H	4CH
名　　称	プログラムの終了	プログラムの常駐終了	プロセスの終了
入力条件	INT 21Hを実行するときにCSセグメントのオフセット0~FFHにPSPがあること 〔一口でいえばCOM型．そうでなくてもスタックを調整して上記のようにすればよい〕	AL←子プロセスの終了コード下1バイト目（リターンコード） cf. ファンクション4DH DX←メモリに常駐されるプログラムのパラグラフ数	AL←子プロセスの終了コード 下1バイト目（リターンコード） cf. ファンクション4DH
出力条件	子プロセスの終了コードの下1バイト目は0に強制される 上1バイトは0	子プロセスの終了コードの上1バイト目は3にされる	子プロセスの終了コードの上1バイト目は^C処理中なら1にそうでなければ0にされる
		子プロセスの終了コードの下1バイトは入力条件のALにされる	

注）致命的エラーでAbort実行して終了するときには，子プロセスの終了コード，上1バイトは2にされる．

図3・22　プログラムの終了関係ファンクション（ファンクション0，31H，4CH）の入出力条件

となるように設定し，INT 22Hの割り込みベクタアドレスへファージャンプします．（INT 22Hを実行するのではない）．このために，INT 22Hのベクタアドレスのことを"プログラムの終了アドレス"というのです．したがってユーザーがプログラムを終了するために，INT 22Hを実行する必要はないのです．また，そのためにINT 22Hを実行してはいけません．

これら3つのファンクションの入出力条件を図**3・22**に，解析フローチャートを図**3・23**に示しますので，詳細はそちらを参照してください．

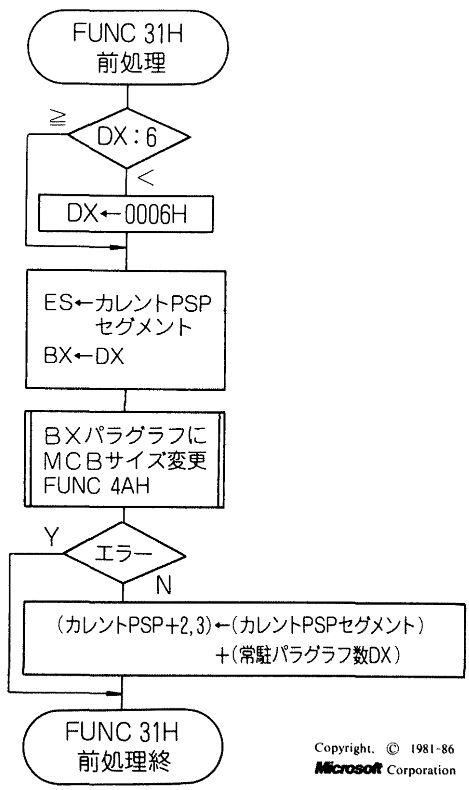


図3・23(b)　プログラム終了関係のファンクションの解析フローチャート（ファンクション31H関係）

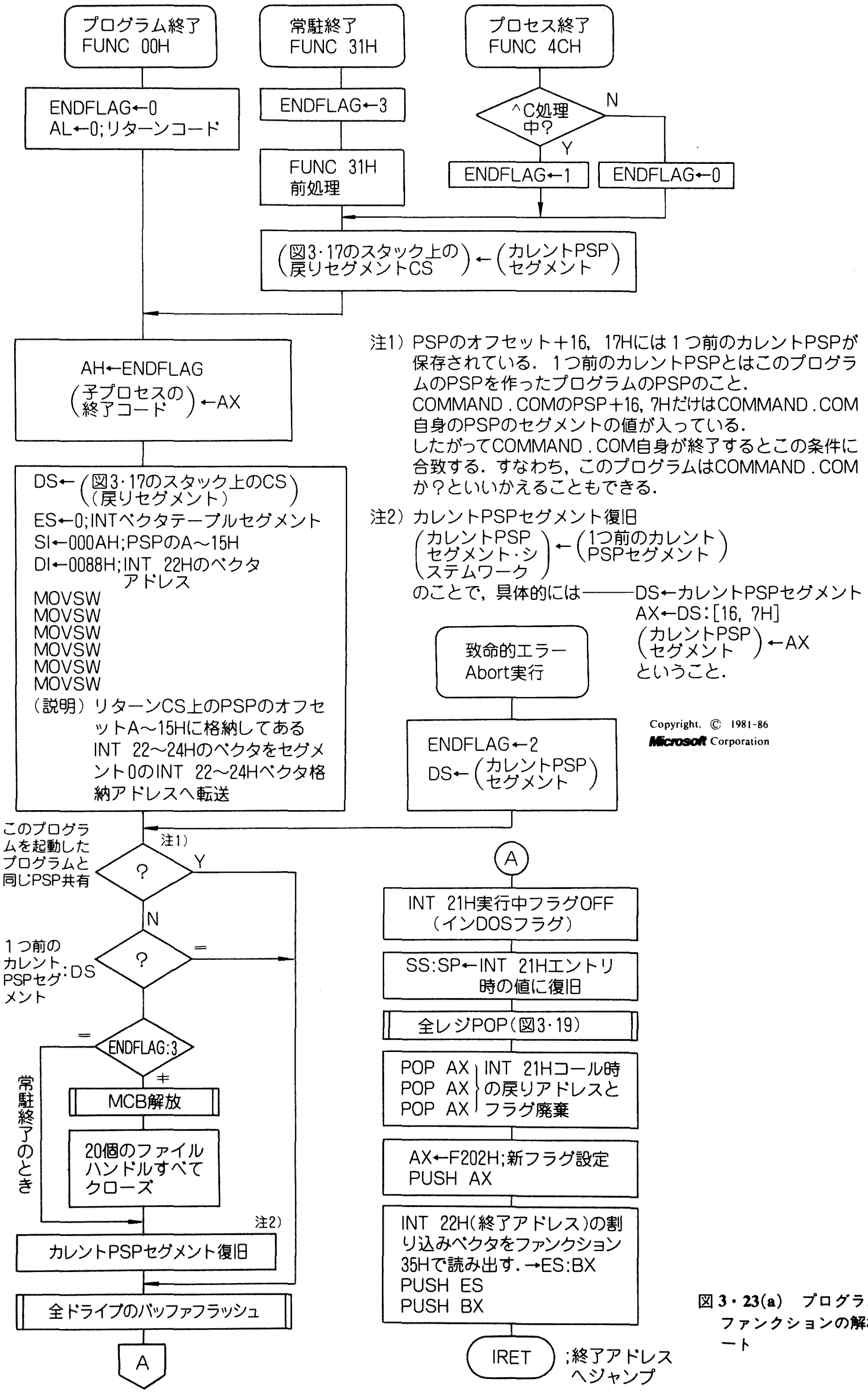


図3・23(a)　プログラム終了関係のファンクションの解析フローチャート

ファンクション01H

キーボード1文字入力（エコーあり）

ファンクション01Hはキーボード(標準入力)から1文字入力し，その文字を画面(標準出力)にも表示します．処理としては簡単で，まず"エコーなしキーボード入力(ファンクション08H)"をコールして，つぎに入力した文字を"1文字CRT出力(ファンクション02H)"で出力しているだけです．

このファンクションの入出力条件と解析フローチャートを**図3・24，25**に示します．

ファンクション番号	01H
名　称	キーボード入力（エコーあり）
入力条件	な　し
出力条件	AL …入力された文字コード CRTに文字出力

図3・24　ファンクション**01H**キーボード入力（エコーあり）の入出力条件

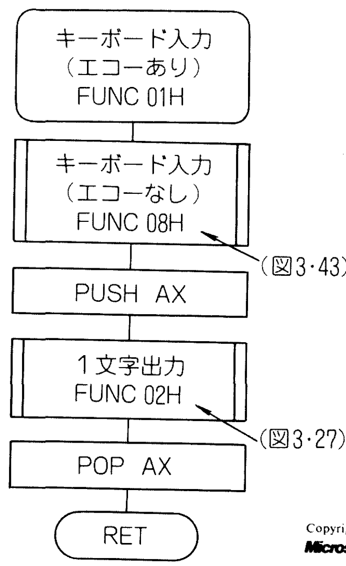


図3・25　ファンクション**01H**キーボード入力（エコーあり）の解析フローチャート

ファンクション02H
1文字CRT出力

DL レジスタで示されるコードを CRT(標準出力)に出力します. 内部的には AL レジスタで示されるコードを出力するエントリも用意されています. 1 文字出力ですが, 漢字は 2 バイトで 1 文字なので, このファンクション 1 回のコールでは漢字は出力できません(正確には 2 回のコールが必要). 正確には 1 バイト出力とでもいうべきです. 4 バイト出力に 1 回, "^C, ^S, ^P, ^N チェック"処理ルーチンを呼び出しています. もし ^P(^N)フラグが ON になっていれば, 標準プリンタ出力(ファイルハンドル 4)にも出力します.

コントロールコード(0~1FH)については, 改行(0DH), 後退(08H), タブ(09H)の処理を実行するようになっています. これらの処理は, MSDOS . SYS 内で独自に持っている X 座標カウンタに対して行われます.

図 3・26 に入出力条件, **図 3・27** に解析フローチャートを示します.

ファイルハンドル・オープンチェック&内部 FCB アドレス取得

図 3・27 などのフローチャートを見ると, "ファイルハンドルはオープンされているか"というような部分がありますが, これはいったいどのような処理をしているのでしょうか? ファイルハンドル番号を示されて, いまそれがオープンされているかどうかをチェックするのは簡単です. **図 3・8** を見ると, PSP の+18~2BH にファイルハンドルと内部 FCB 番号対応表があることは示しました. オープンされていない FH は, 対応する内部 FCB 番号に FFH が書かれています. オープンされている FH は対応する内部 FCB 番号が書かれていますので, それから内部 FCB のアドレスを計算することができます. この計算は**図 3・28** のフローチャートのほうを参照してください. こうして内部 FCB のアドレスがわかれば, あとは FCB を使ったアクセスに還元して処理することもできるのです.

ファンクション番号	2
名　　称	1文字 CRT 出力
入力条件	DL…出力したい文字コード
出力条件	な　　し

図 3・26　ファンクション 2　1 文字 CRT 出力　入出力条件

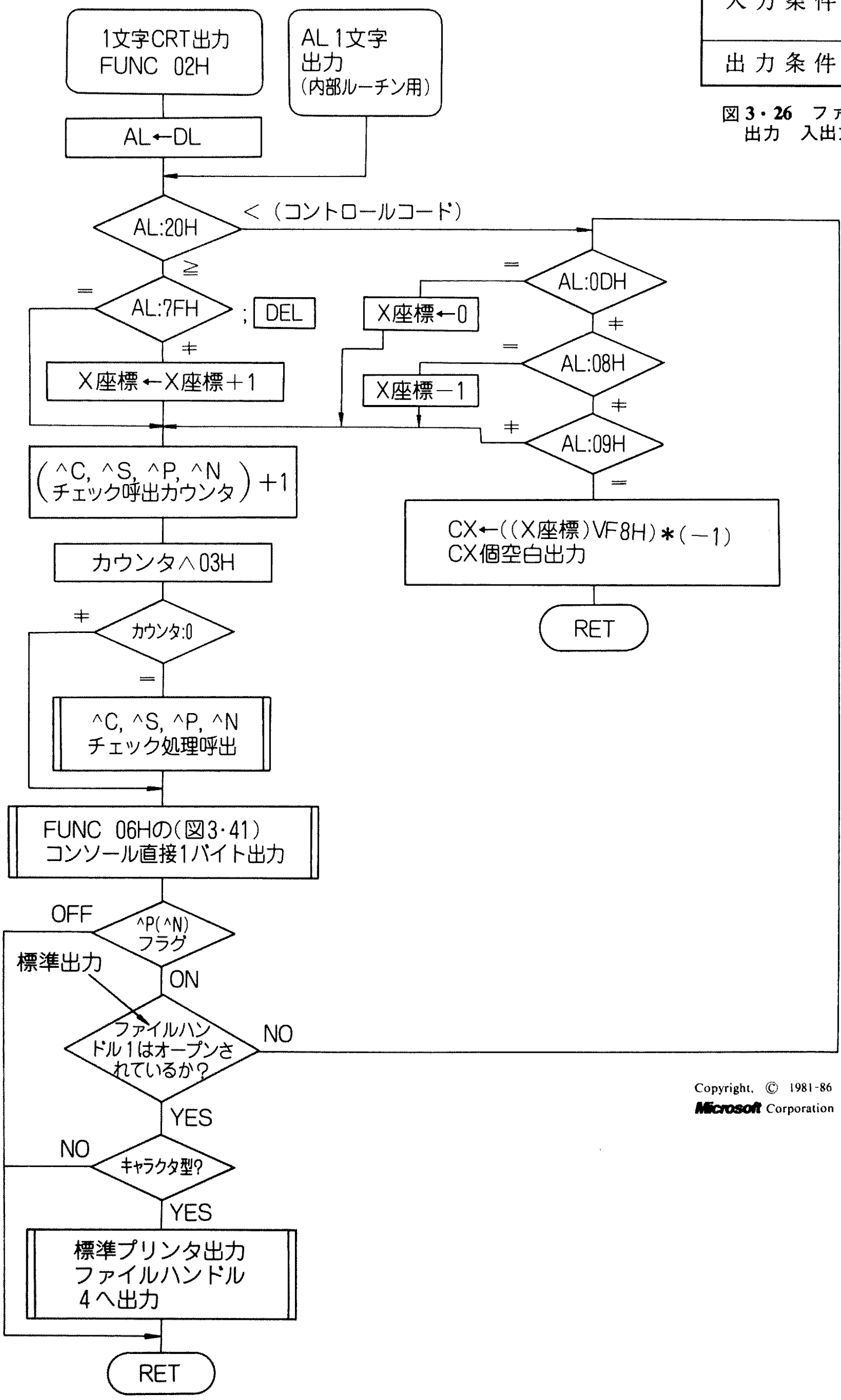


図 3・27　ファンクション 2　1 文字 CRT 出力　解析フローチャート

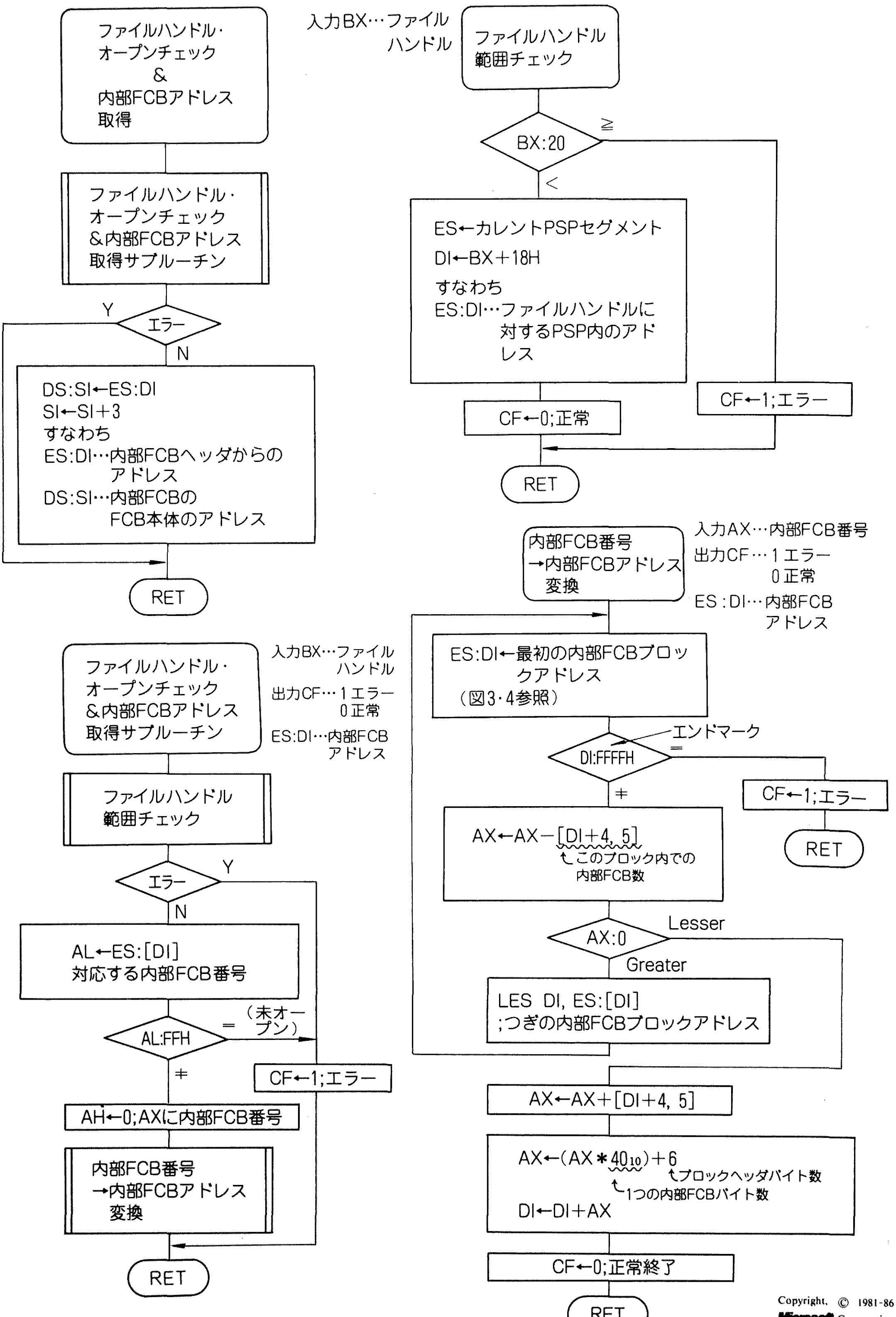


図3・28 ファイルハンドルオープンチェック&内部FCBアドレス取得

ファンクション03H
標準補助入力（ウエイトあり）

補助入力とは，AUX からの入力のことです．AUX は RS-232C に接続されています．標準補助入力はファイルハンドル3に割り当てられており，ファイルハンドル3のオープン状態のチェック&内部 FCB アドレス取得により，FCB 内のデバイスヘッダアドレス(キャラクタ型デバイスの場合は FCB＋19～1CH がデバイスヘッダアドレスとなっている)を使って，"I/O リクエストコール"を実行します．I/O リクエストコールの"キュー非破壊入力"を実行し，入力があるまで待ち，入力があれば I/O リクエストコールの"リードデータ"で入力データを読み込みます．

やはりこのファンクションもエントリ付近で，"^C，^S，^N，^P チェックルーチン"を呼び出しています．

図 3・29 に入出力条件，**図 3・30** に解析フローチャートを示します．

ファンクション番号	3
名　　称	標準補助入力
入力条件	な　　し
出力条件	AL…標準補助入力から入力されたコード

図 3・29　ファンクション 3　標準補助入力
入出力条件

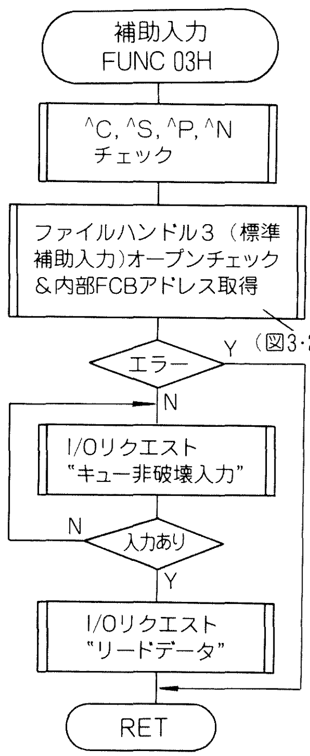


図 3・30　ファンクション 3　標準補助入力
解析フローチャート

I/O リクエストコール呼び出しの実際

① キャラクタ型I/O リクエストコール

図3・30(ファンクション3のフローチャート)でI/O リクエストコールの"キュー非破壊入力"と"リードデータ"が出てきましたが，MSDOS.SYS 内ではこれらをどのように実行しているのでしょうか？

リードデータ(コマンド4)，キュー非破壊入力(コマンド5)，ライトデータ(コマンド8)，出力用ステータスチェック(コマンド10)，入力バッファフラッシュ(コマンド7)はひとまとめにして，それぞれAH レジスタに0～4の番号を入れてCALL するような処理ルーチンを作成しています．ブロック型デバイスについても0～2(リードデータ，キュー非破壊入力，ライトデータ)をサポートしていますが，本来I/O リクエストコールではブロック型のキュー非破壊入力をサポートしていないこともあって直接I/O リクエストコールをせず，MSDOS.SYS のファイル管理に即すためにランダム入出力ルーチンを使って実現しています(ブロック型のI/O リクエストコールはあとで述べる)．

図3・31(a)～(c)に，この"I/O リクエストコール用サブルーチン"の解析フローチャートを示します．**図3・31(c)**を除いてこのルーチンは，おもにキャラクタ型のデバイスを対象に書かれていて

- コンソール直接入出力(ファンクション06H)……コマンド0～3を利用
- コンソール直接入力　(ファンクション07H)……コマンド0～3を利用
- 標準補助入力　　　　(ファンクション03H)……コマンド0，1を利用
- エコーなしキーボード入力(ファンクション08H)……コマンド0を利用
- タイプアヘッドバッファを空にしてキーボード入力(ファンクション0CH)……コマンド4を利用
- I/O コントロール(ファンクション44H のコマンド6，7)……コマンド1，3を利用
- ^P，^N，^C，^S チェックルーチン　……　コマンド0，1を利用

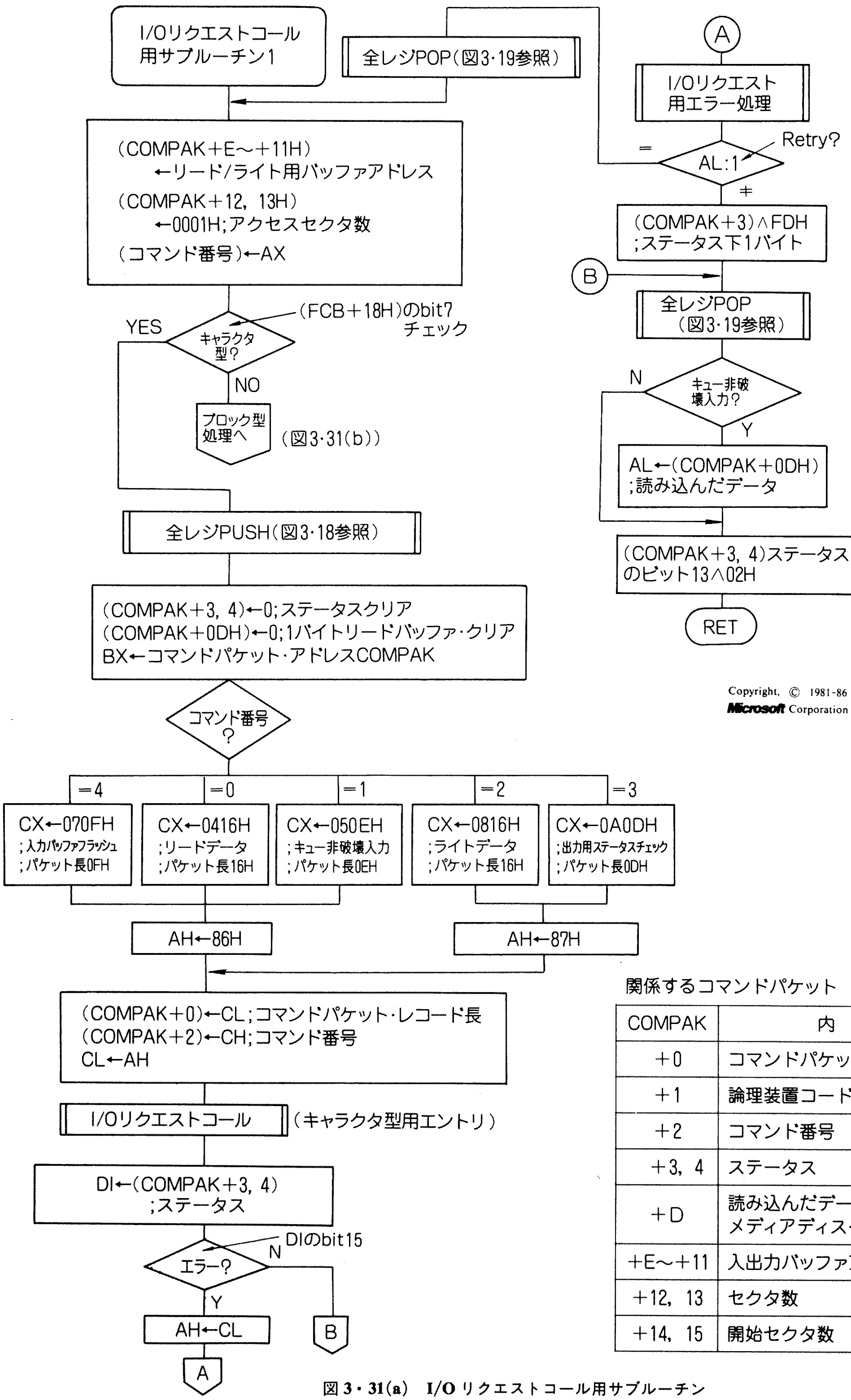
からCALL されています．くわしい使用のされかたは，それぞれのファンクションコールの解析のほうを参照してください．

② ブロック型I/O リクエストコール

図3・31のルーチンはおもにキャラクタ型のI/O リクエストコールに利用されましたが，ブロック型は**図3・31(c)**のI/O リクエストコールを使っています．このようなブロック型アクセスはFAT やディレクトリの読み出し，メディアチェック&BPB よりDPB 作成(更新)，時刻・日付のリード/ライト，I/O コントロール(ファンクション44H のコマンド2～5)で使用されており，またINT 0 除算エラー処理(エラーメッセージ出力)，^C チェック(INT 25H 直接ディスクリード，INT 26H 直接ディスクライト)などの直接デバイスドライバへ，アクセスが必要なエラートラップ関係の処理でも使われています．

I/O リクエストコールにはデバイスヘッダアドレスが必要ですが，キャラクタ型はおもにFCB の＋19H～1CH を利用しますが，ブロック型では(FCB＋19H～1CH は最初のクラスタ番号やカレントディレクトリのクラスタ等が記録されていて，FCB にはデバイスヘッダアドレスが記録されていないので) DPB の＋12H～15H を利用しています．

また，データリード/ライトは頻繁に使用するので，リード/ライト用コマンドパケットを作成するルーチンを設けています(**図3・32**参照)．このルーチンをCALL したあと**図3・31(c)**の"I/O リクエストコール"ルーチンをCALL して，デバイスへのリード/ライトを実行するようになっています．



関係するコマンドパケット

COMPAK	内　容
+0	コマンドパケットレコード長
+1	論理装置コード
+2	コマンド番号
+3, 4	ステータス
+D	読み込んだデータ(非破壊入力) メディアディスクリプタ
+E〜+11	入出力バッファアドレス
+12, 13	セクタ数
+14, 15	開始セクタ数

図3・31(a)　I/O リクエストコール用サブルーチン

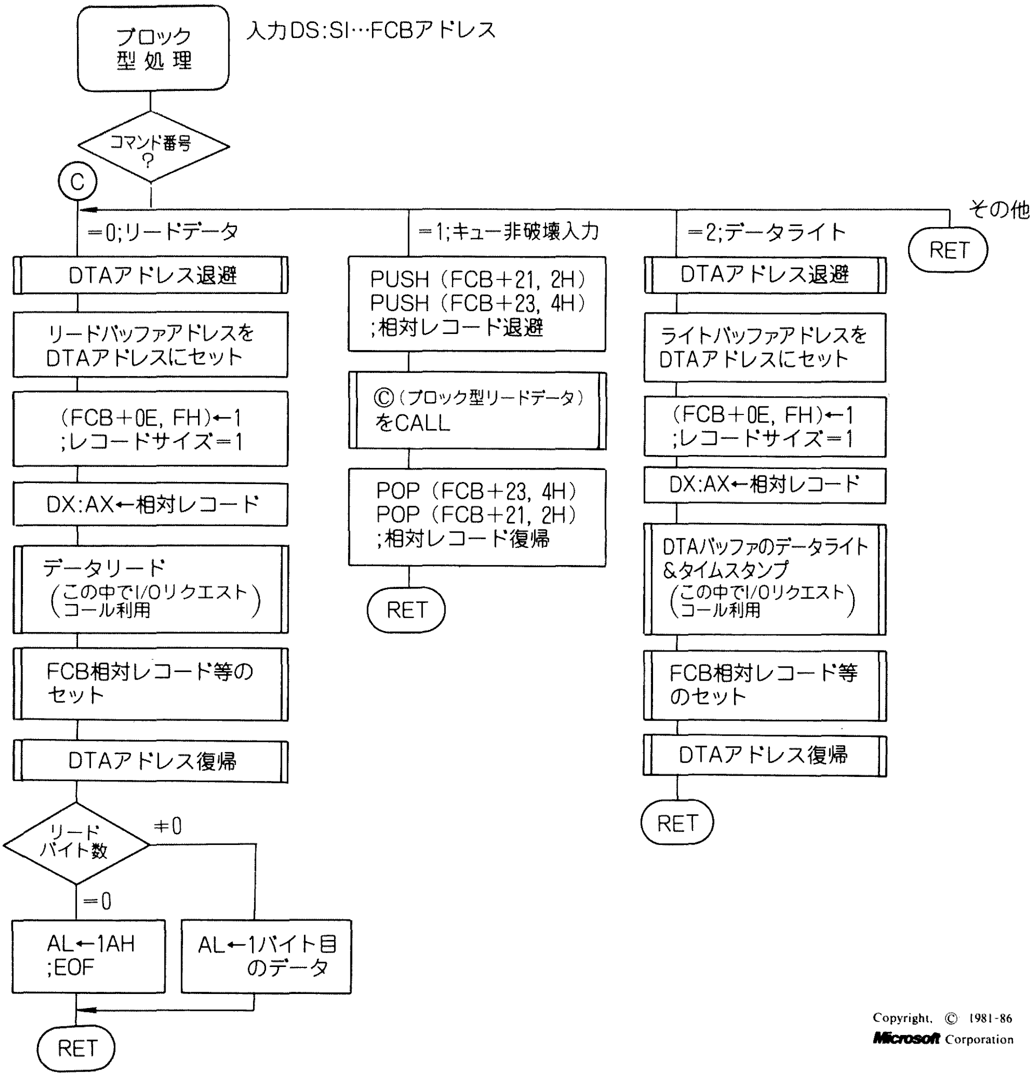


図 3・31(b) I/O リクエストコール用サブルーチン

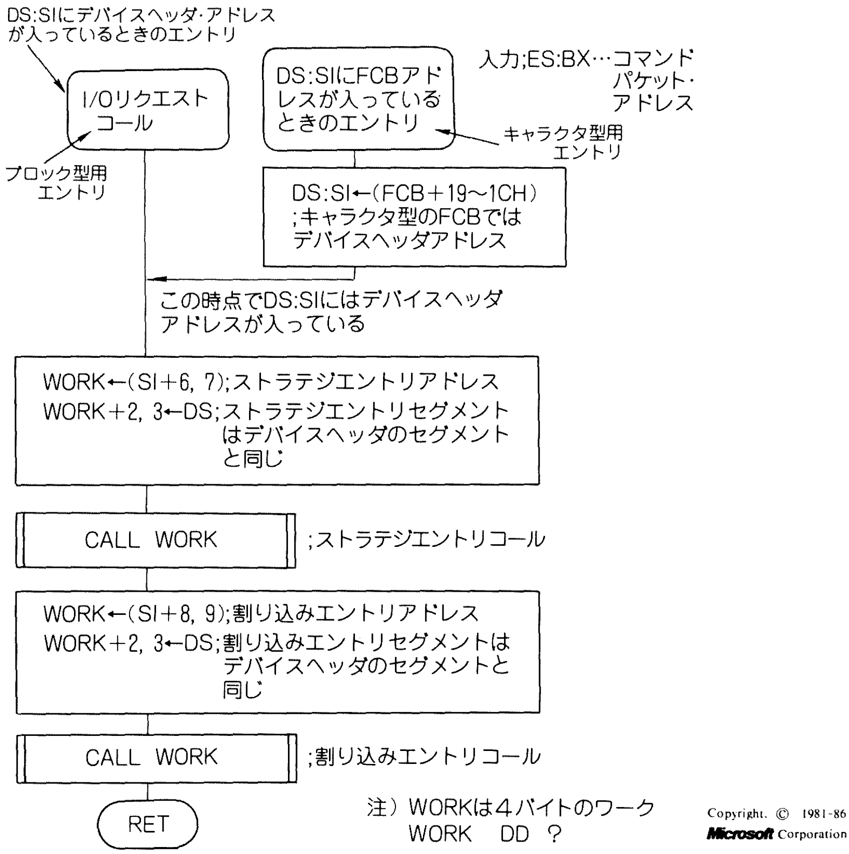


図 3・31(c)　I/O リクエストコール用サブルーチン

I/O リクエストコール用エラー処理と INT 24H

図 3・31(a)で I/O リクエストコールを実行したのち，エラー(ステータスの bit15=1)なら"I/O リクエストコール用エラー処理ルーチン"を CALL していますが，このエラー処理の中で致命的エラー処理 INT 24H を実行しています．INT 24H の処理ルーチン自体は MSDOS . SYS にはなく，COMMAND . COM か外部コマンドが処理するようになっています．この処理の流れを示すと**図 3・33** のようになります(ただし外部コマンドで INT 24H ベクタを変更しなければ通常 COMMAND . COM 内の処理ルーチンとなります)．

ただし，MSDOS . SYS から見た INT 24H の入出力条件というものはきめられています．すなわち，MSDOS . SYS が INT 24H を実行するときは

① DI の下位 8 ビットに I/O リクエストコールからリターンしたエラー番号
② BP : SI にエラーの発生したデバイスドライバのデバイスヘッダアドレス
③ INT 21H 実行中フラグ(IN-DOS フラグ)を OFF
④ INT 24H 致命的エラー処理実行中フラグを ON
⑤ SS : SP には INT 21H にエントリしたときのコマンドで使っていたスタックのアドレス
⑥ AH にエラー発生セクタの箇所番号(**図 3・34(a)**参照)
⑦ AL にエラー発生ドライブ番号

この中で④の INT 24H 実行中を表わすフラグを ON にするには，INT 24H 処理ルーチンの中で INT

21H を利用していた場合を考慮してのことです．もし，INT 24H 処理ルーチンから INT 21H を実行して I/O リクエストコールでふたたびエラーが発生したら， INT　24H を実行せず Ignore　処理を選択したようにしてリターンします．これによって INT　24H 処理ルーチンは，リエントラント，リカーシブに作る必要はなくなります．

MSDOS . SYS は逆に INT　24H 処理ルーチンからのリターン値として AL レジスタに

AL：　0……　Ignore
　　　1……　Retry
　　　2……　Abort

を期待します．Abort(AL＝2)のときは，**図 3･23(a)**で示されるようにすぐにプリンタ終了処理をします．Ignore(AL＝0)と Retry(AL＝1)のときは，そのまま I/O リクエストコール用エラー処理を CALL したところへリターンします．

I/O リクエストコール時のエラー処理の解析フローチャートを**図 3･34(a)(b)**に示します．これらのことを知っていれば，自分用の INT 24H 処理ルーチンを作ることができます．INT 24H 実行時に BP：SI にデバイスヘッダアドレスがセットされていますので，これを利用することもできます．

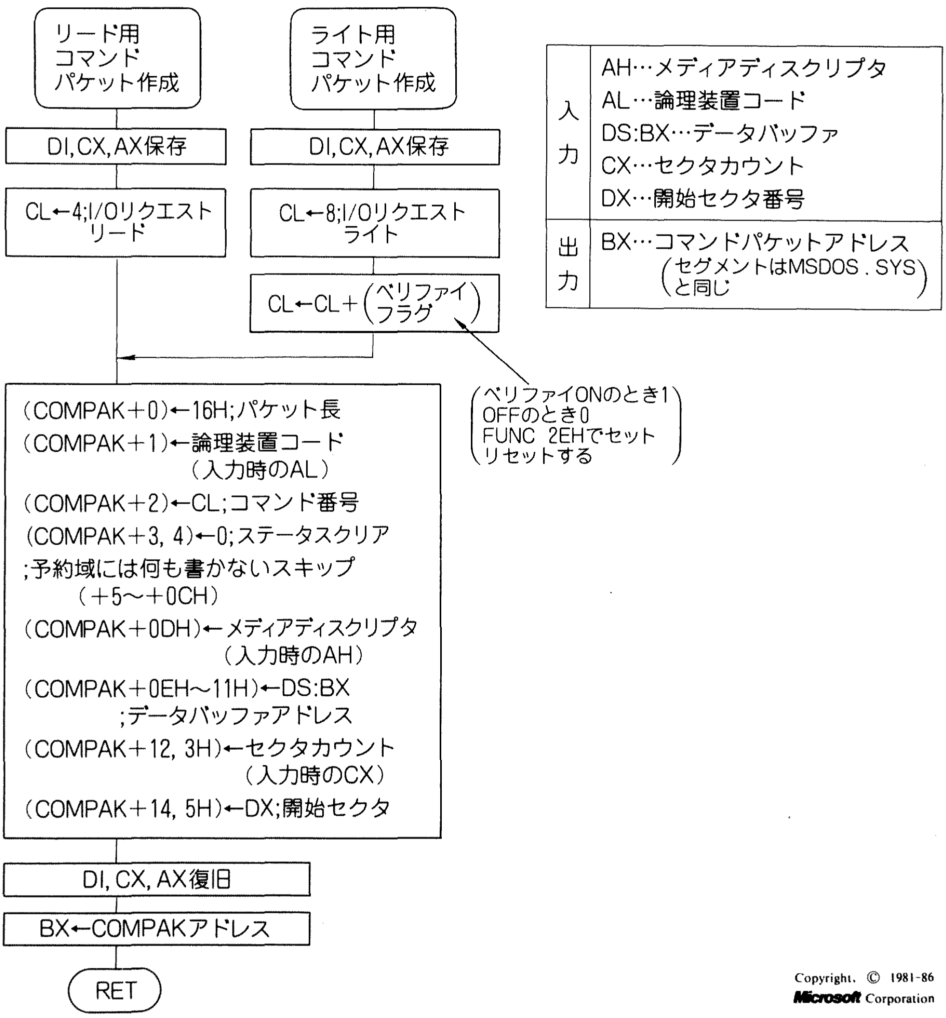
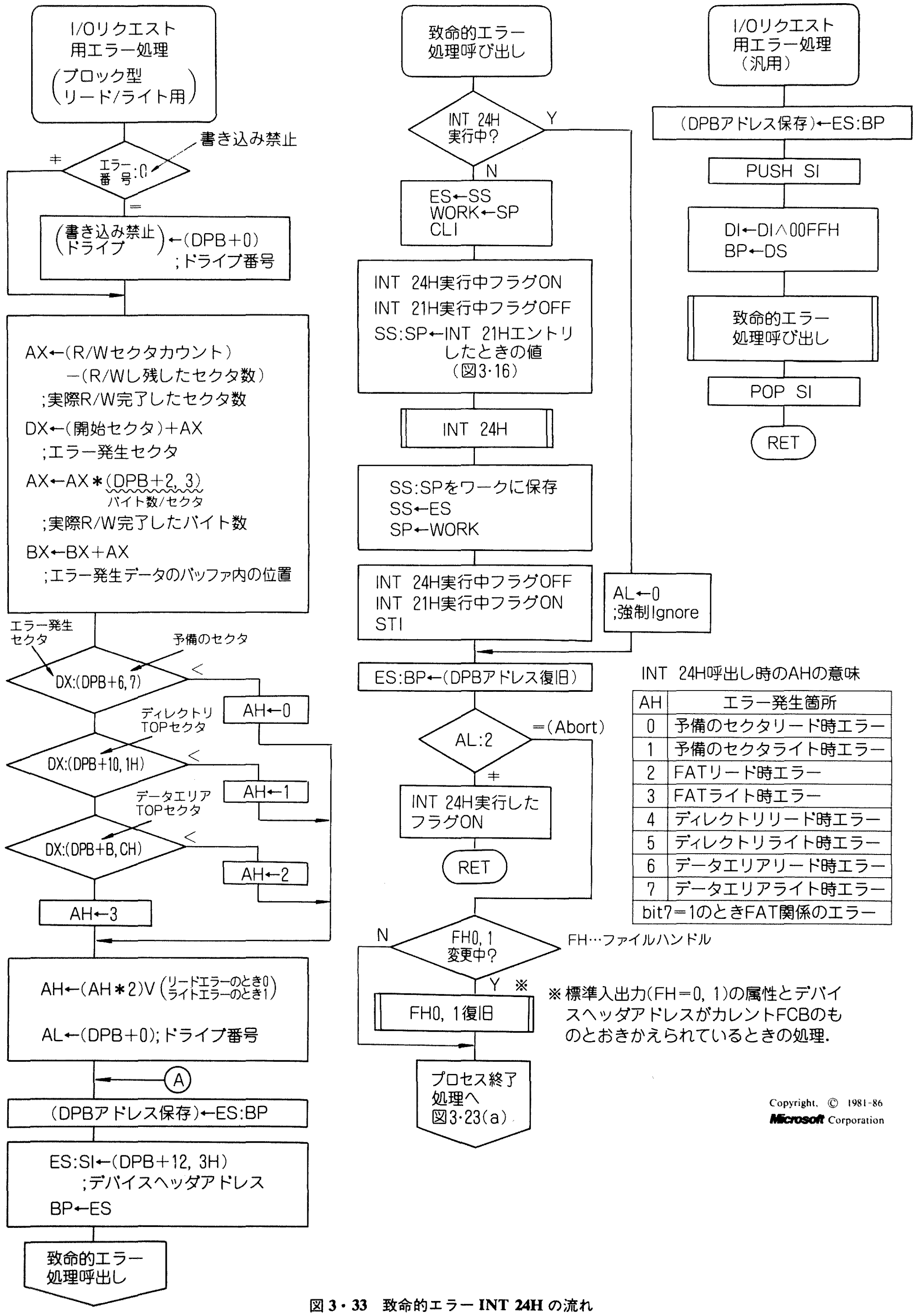


図 3・32　リード/ライト用コマンドパケット作成ルーチン（I/O リクエストコール用）



INT 24H呼出し時のAHの意味

AH	エラー発生箇所
0	予備のセクタリード時エラー
1	予備のセクタライト時エラー
2	FATリード時エラー
3	FATライト時エラー
4	ディレクトリリード時エラー
5	ディレクトリライト時エラー
6	データエリアリード時エラー
7	データエリアライト時エラー
bit7=1のときFAT関係のエラー	

図 3・33　致命的エラー INT 24H の流れ

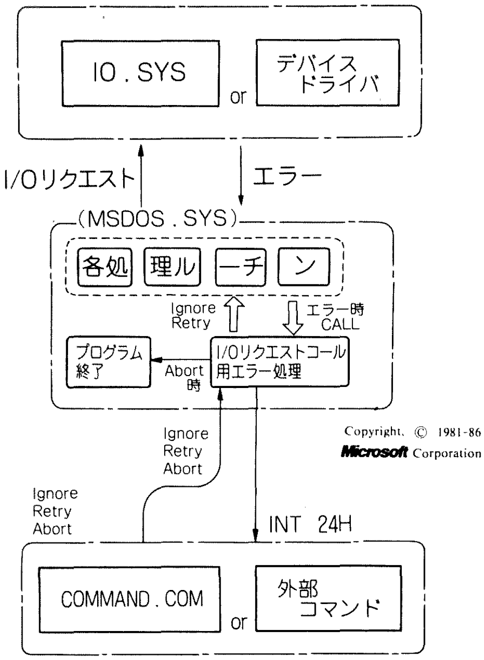


図3・34(a) I/Oリクエストコール時致命的エラー処理(フローチャート)

	I/Oリクエスト用エラー処理(ブロック型 リード/ライト用)	I/Oリクエスト用エラー処理(汎用)
入力	AX …エラー番号 BX …入出力バッファアドレス CX …R/W 残しセクタ DX …開始セクタ DI …セクタカウント ES : BP…エラー発生 DPB	DI …エラー番号 DS : SI…デバイスヘッダ ES : BP…エラー発生 DPB
出力	AL …0 Ignore 1 Retry (2 Abort 時はすぐに終了処理実行)	

図3・34(b) I/Oリクエストコール時致命的エラー処理(入出力条件)

^C チェック&^C 処理と INT 23H

^C(コントロールC)は，CTRL キーを押しながら'C'キーを押すことです．STOP キーを押しても同様のコードが送られます．INT 21H ファンクション 33H で^C チェックフラグを ON にしておけば，INT 21H ファンクションコールを実行するたびに^C チェック&処理ルーチンが CALL されます．また，ファンクション 01～0DH の入出力関係処理では^S，^P，^N チェック&処理ルーチンといっしょに CALL されます．

^C がセンスされると CRT に"^C"と表示され，INT 23H(^C による中断処理)が実行されます．くわしくは，**図 3・35** の解析フローチャートを参照してください．INT 23H(^C による中断)を実行するときに MSDOS . SYS は，SS：SP(スタック)を INT 21H エントリ時の値にもどし，全レジスタ POP し INT 21H 実行中フラグも OFF にするので INT 23H 処理ルーチンではそのまま IRET せずに処理の実行権をとってしまうこともできますが，つぎのように MSDOS . SYS 内の^C 処理の続きへ IRET することもできます．すなわち，AH レジスタにファンクションコードを入れ，必要なら他のレジスタに値をセットし，INT 21H をする代わりに IRET することもできます．こうすると**図 3・35** の解析フローチャートを見てもわかるように，スタックが変化していなかったら INT 21H ファンクションコールエントリへエントリしてファンクションを実行してしまいます．この機能は INT 21H のなんらかのファンクションを実行中に^C をかけた場合，^C の処理(これはいろいろ考えられ，たとえば画面に"Don't Stop Me"など表示して)をしたのち，ふたたび中断されたファンクションを再実行するなど，さまざまな応用が考えられます．

^C，^S，^P，^N チェック処理

^C については"^C チェック&^C 処理"ルーチンを CALL するだけですから，そちらのコラムのほうを参照してください．^S をセンスすると，つぎのキー入力があるまでループします．^P と^N はまったく同じで，画面出力と同時にプリンタにも出力するかどうかのフラグを，トグルで切り換えます．^P(^N)をセンスするたびにフラグのメモリを NOT して，00H⟷FFH をトグルで切り換えているのです．

このチェックルーチンは

① 1 文字出力(ファンクション 02H)
② 標準補助入力(ファンクション 03H)
③ 標準補助出力(ファンクション 04H)
④ 標準プリンタ出力(ファンクション 05H)
⑤ キーボード入力(ファンクション 08H)
⑥ キーボードステータスチェック(ファンクション 0BH)

の要所要所で CALL されています．すなわち，これらは CON，AUX，PRN 関係の入出力ファンクションなので，CON，AUX，PRN 入出力中は^C，^S，^P，^N の機能が使えるというわけです．

解析フローチャートを**図 3・36** に示します．INT 28H は通常 IRET 命令を指していて，なにも処理はしていませんが，PRINT. COM を起動すると PRINT. COM 内のプリンタ出力ルーチンへ INT 28H ベクタが書き換えられ，**図 3・36** などのフローチャートからわかるように，CON のキー入力待ちのときにプリンタ出力処理を行うようになっています．PRINT . COM はタイマ割り込みと上に述べた両方の方法でプリンタに出力しています．

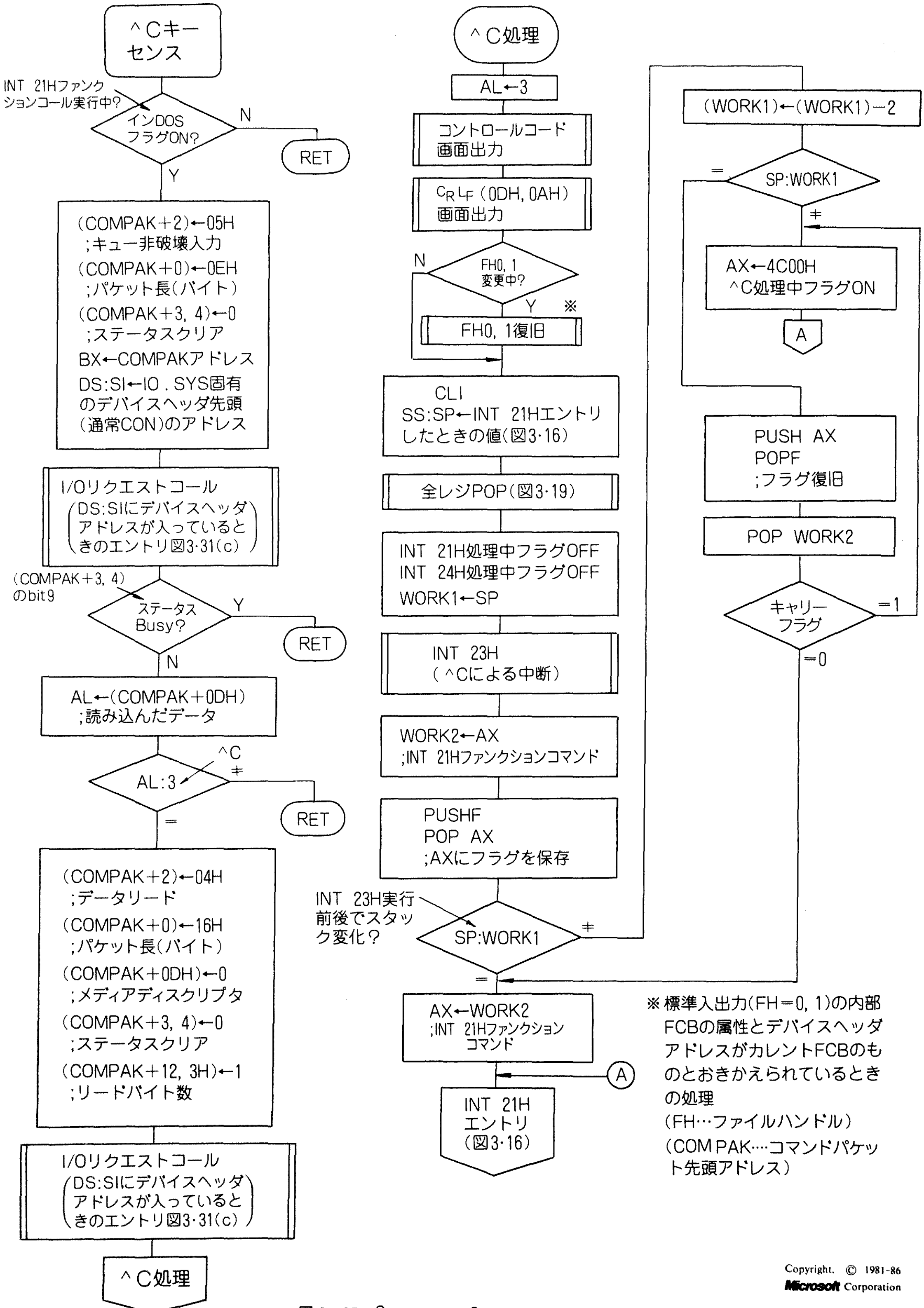


図3・35 ^Cセンス&^C処理解析フローチャート

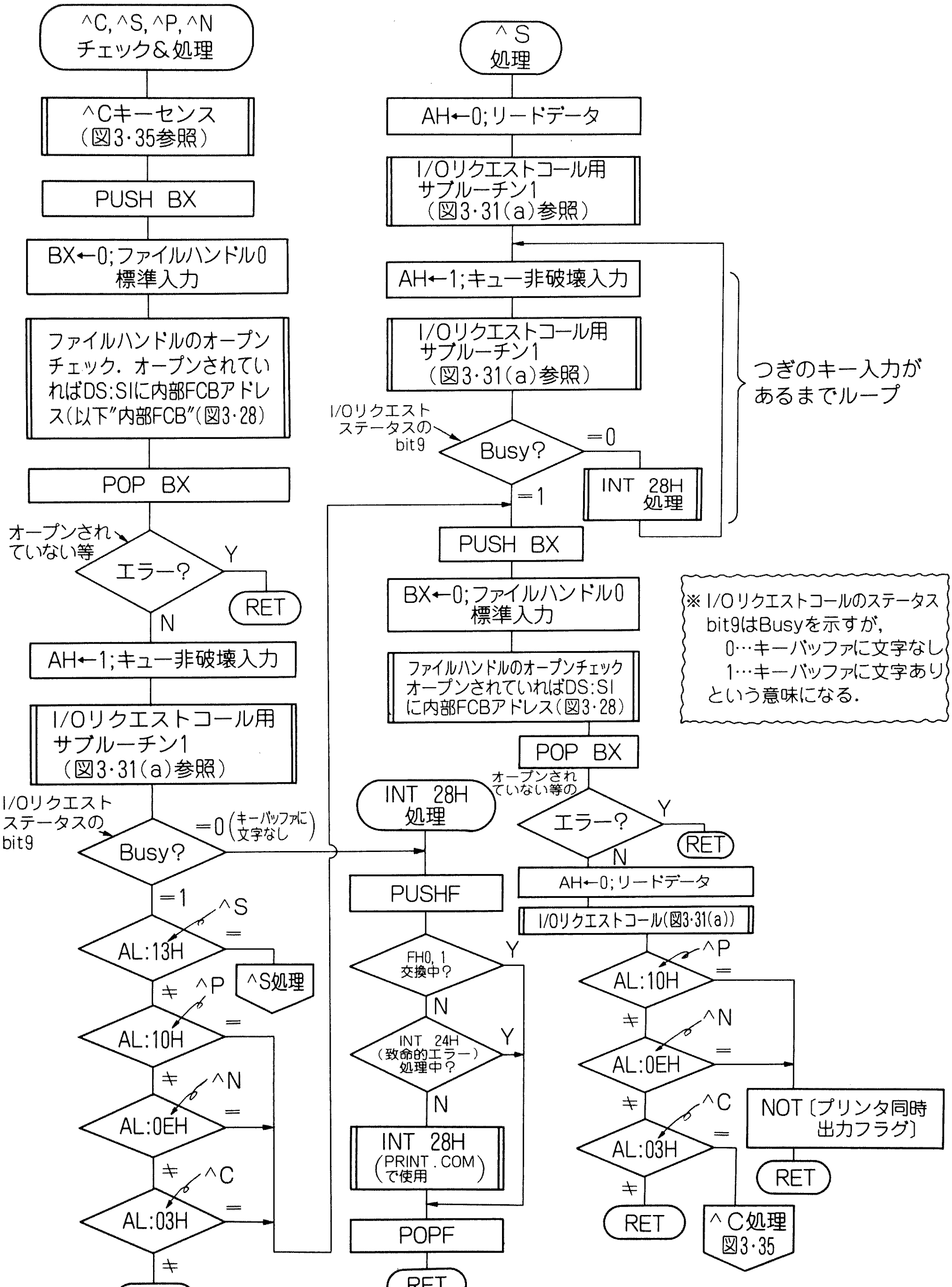


図3·36 ^C, ^S, ^P, ^N チェック&処理 解析フローチャート

ファンクション04H
標準補助出力

ファイルハンドル3(標準補助出力)へ1バイト出力することで実現しています．解析フローチャートを**図3・37**に示しますが，ファンクション05H(標準プリンタ出力)も同様の方法をとっていますので，ファンクション05Hと共通のフローチャートとなっています．

入出力条件を**図3・38**に示します．

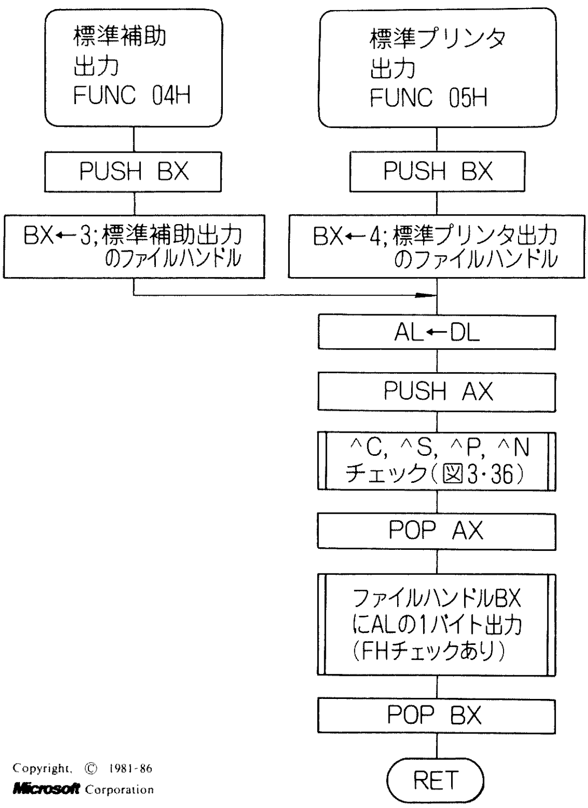


図3・37 ファンクション4 (標準補助出力)
ファンクション5 (標準プリンタ出力)
解析フローチャート

ファンクション番号	4，5
名　　称	4…標準補助出力 5…標準プリンタ出力
入力条件	DL …出力したい文字コード
出力条件	な　　し

図3・38 ファンクション4 (標準補助出力)
ファンクション5 (標準プリンタ出力)
入出力条件

ファンクション05H

標準プリンタ出力

ファイルハンドル4(標準プリンタ出力)へ1バイト出力することで実現しています．解析フローチャートを**図3•37**に，入出力条件を**図3•38**に示します．ここで”ファイルハンドルBXにALの1バイト出力”というサブルーチンをCALLしていますが，この処理を**図3・39**に示します．

この処理には，最初にファイルハンドルのオープンチェックをするエントリとしないエントリの2つがあります．ファンクション4, 5で使っているのはファイルハンドルのオープンチェックをするエントリです．オープンチェックをしないほうのエントリは，コンソール直接入出力(ファンクション6)の出力処理で使用しています．

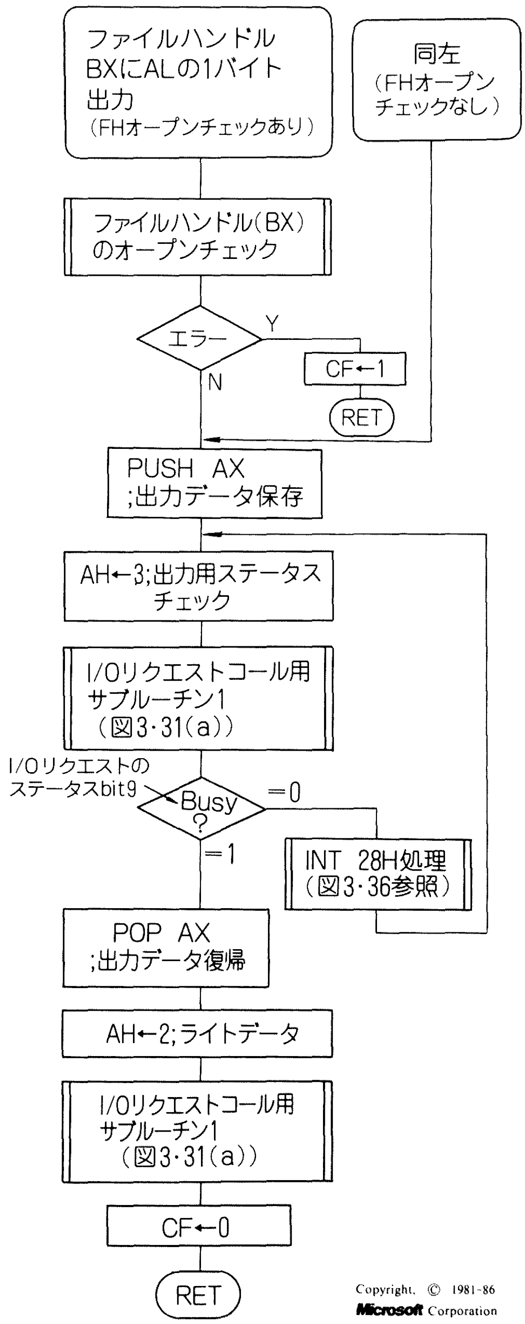


図3・39 ファイルハンドルによる
1バイト出力フローチャート

ファンクション06H
コンソール直接入出力

ファンクション07H
コンソール直接入力

はじめにDLレジスタのコードを調べて，FFHなら1バイト出力ルーチンへ，それ以外なら1バイト入力ルーチンへ飛びます．1バイト出力ルーチンはファンクション02HからもCALLされます．ファンクション07Hはファンクション06Hの1バイト入力ルーチンの途中からエントリします．ファンクション06Hとファンクション07Hの入力の違いは，リターン時のゼロフラグに意味があるかないかです．ファンクション07Hのゼロフラグには意味がありませんが，ファンクション06Hのゼロフラグには

0…… (NZ) 入力なし

1…… (Z)　入力あり

という意味があります．リターン時のゼロフラグを0，1にセットするには，スタック中に待避されているフラグレジスタの値を直接扱っています．

入力にはファイルハンドル0，出力にはファイルハンドル1を使っています．ただし，出力はINT 29Hを実行することによって行っています．

入出力条件を**図3・40**，解析フローチャートを**図3・41**に示します．

<table>
<tr><td>ファンクション番号</td><td colspan="2">6</td><td>7</td></tr>
<tr><td>名　　称</td><td colspan="2">コンソールからの直接入出力</td><td>コンソール直接入力</td></tr>
<tr><td>入力条件</td><td>DL ← FFH
(入力)</td><td>DL ← FFH
以外
(出力)</td><td>な　し</td></tr>
<tr><td>出力条件</td><td>ZF…1
入力なし
(AL=0)
ZF…0
入力あり
ALにそのコード</td><td>な　し</td><td>AL …入力されたコード</td></tr>
</table>

図3・40　ファンクション**6**（コンソール直接入出力）
ファンクション**7**（コンソール直接入力）入出力条件

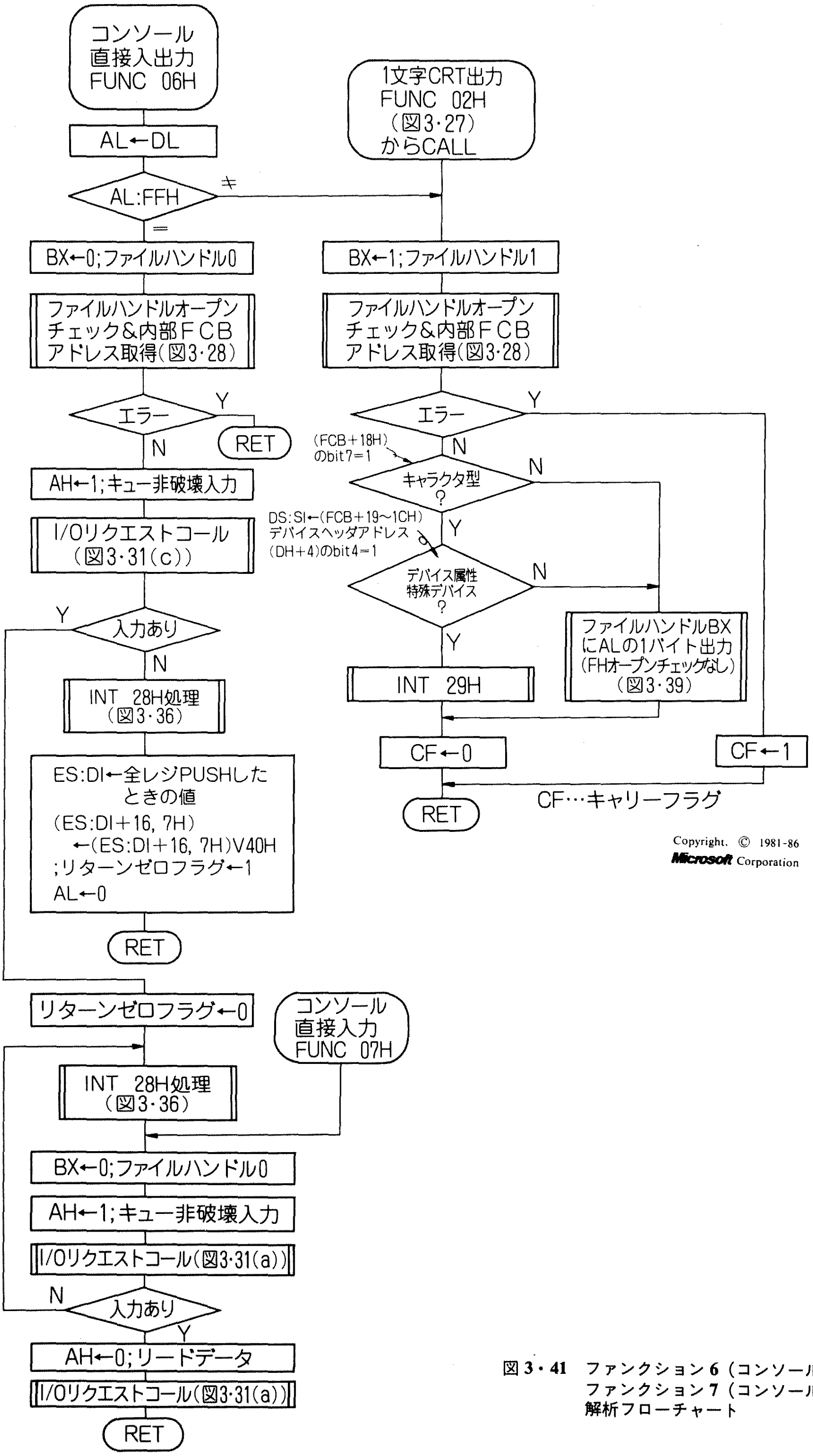


図3・41 ファンクション6(コンソール直接入出力)
ファンクション7(コンソール直接入力)
解析フローチャート

ファンクション08H

キーボード入力（エコーなし）

"^C, ^S, ^P, ^N チェック"ルーチン（**図 3・36** 参照）を CALL して，なにか入力があれば I/O リクエストコールのリードデータを CALL して，キー入力コードを得ます．それまではループします．"^C, ^S, ^P, ^N チェック"ルーチンのリターン条件は

ゼロフラグ …… 1（Z）　入力なし
　　　　　　　0（NZ）　入力あり

です．

図 3・42 に入出力条件，**図 3・43** に解析フローチャートを示します．

ファンクション番号	8
名　称	キーボード入力（エコーなし）
入力条件	な　し
出力条件	AL…入力されたコード

図 **3・42**　ファンクション **08H**（キーボード入力）入出力条件

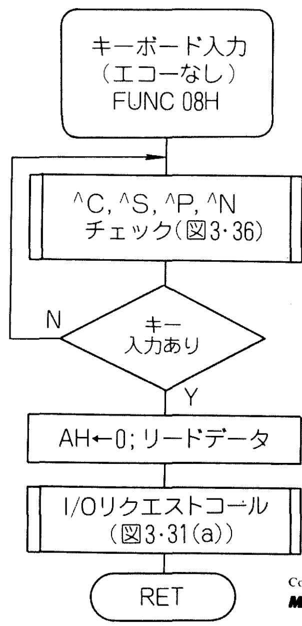


図 **3・43**　ファンクション **08H**（キーボード入力）解析フローチャート

ファンクション09H
文字列出力

1バイト出力には，ファンクション02Hの内部ルーチン用エントリ（**図3・27**参照）を用いています．

入出力条件は**図3・44**，解析フローチャートは**図3・45**に示します．

ファンクション番号	9
名　　称	文字列出力
入力条件	DS：DX…出力したい文字列を格納したバッファアドレス
出力条件	な　　し

図3・44　ファンクション**09H**　文字列出力　入出力条件

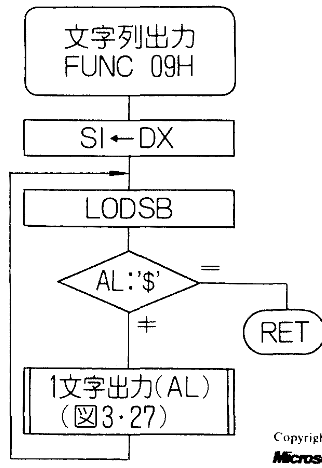
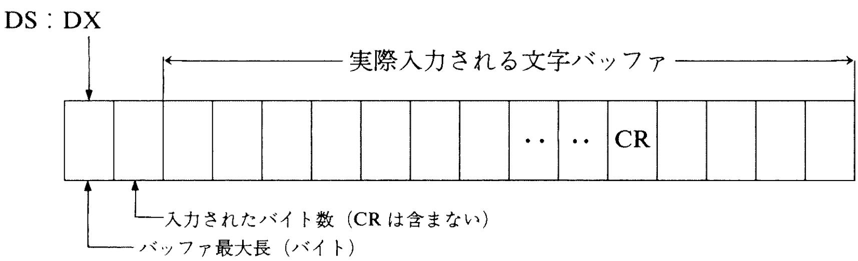


図3・45　ファンクション**09H**　文字列出力
解析フローチャート

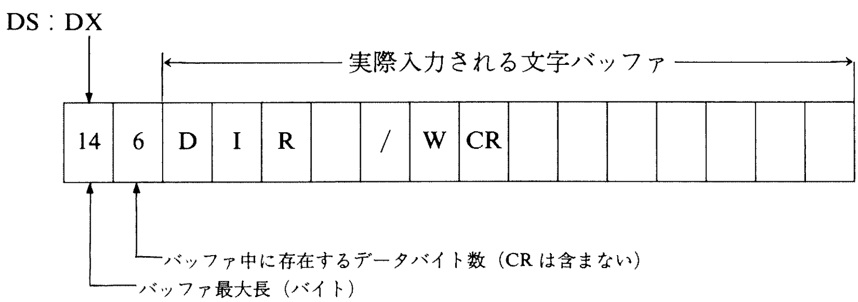
ファンクション0AH
バッファードキーボード入力

キーボード入力にはファンクション08H（エコーなしキーボード入力，**図3・43**参照）を使っています．入力された文字はエコーしますが，そのための出力には，ファンクション02Hの内部ルーチン用エントリ（**図3・27**参照）を用いています．

ファンクション0AHのいちばんのポイントは，”テンプレート機能”です．テンプレートとは，ファンクション0AH呼び出し時に入力バッファに文字列をセットしておけばその文字列がテンプレート機能によって再利用されるため再入力が不要になる，というものです．入力用バッファは



という形式をしています．ここで，通常”入力されたバイト数”はMSDOS．SYSの処理ルーチン側で実際に入力されたバイト数がセットされますが，ファンクション0AHエントリ時にバッファ内に文字列をおき，”入力されたバイト数”にそのバイト数をセットすればそれがテンプレートの文字列となります．たとえばつぎのようです．



とセットしてファンクション0AHを実行すれば，”DIR　/W”をテンプレートのもとの文字列として使えます．

テンプレート処理をするためにシステム内にバッファを作り，テンプレートユーザーバッファからの文字列を入力された順にこのバッファに溜め，リターンキー（0DH）が押されてからDS：DXで指定されたユーザーのバッファへ転送するようになっています（**図3・46**参照）．したがって，前にファンクション0AHを使ってキー入力したバッファをクリアせずに使えば，以前入力した文字列

をテンプレートとして使えることになります.

図3・47に入出力条件, **図3・48（a）～（d）**に解析フローチャートを示します.

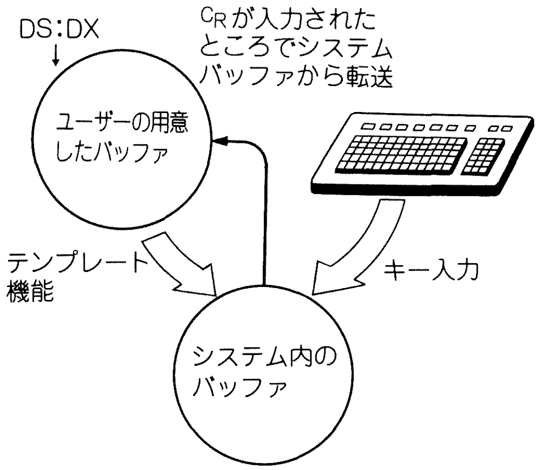


図3・46　バッファード入力（テンプレート付）のデータの流れ

ファンクション番号	0AH
名　　称	バッファードキーボード入力
入力条件	DS:DX…入力バッファアドレス ├── m ──┤ (D から 最後の枠まで) | m | n | D | I | R | CR | | | ├── n ──┤ (D から R まで) m…バッファ最大長（バイト） n…テンプレートバイト数（CR を含まない）
出力条件	○上記バッファに入力された文字列格納 ○上記バッファの n に入力されたバイト数（最後の CR は含まない）

図3・47　ファンクション 0AH　バッファードキーボード入力　入出力条件

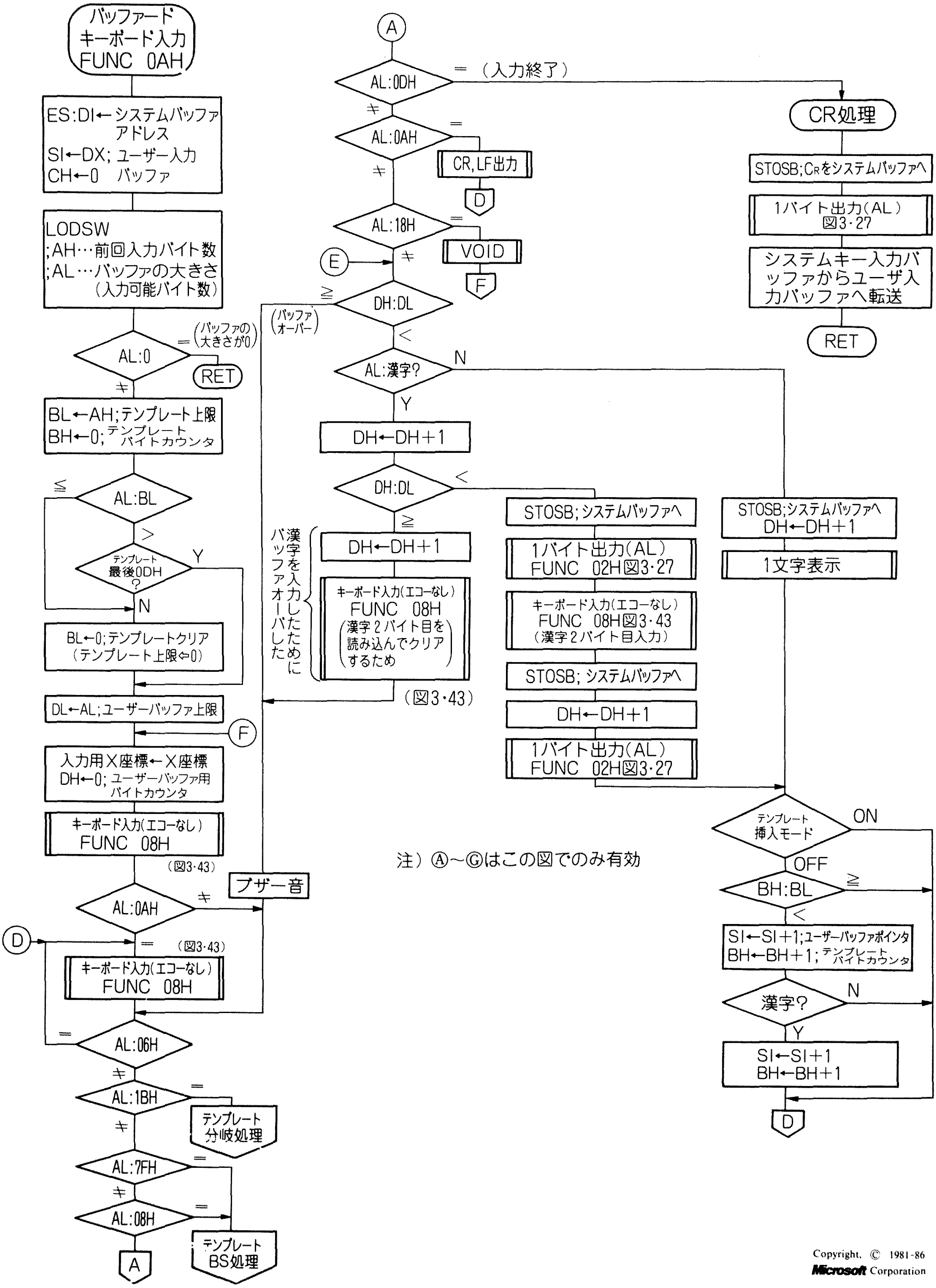


図3・48(a) ファンクション0AH バッファードキーボード入力解析フローチャート

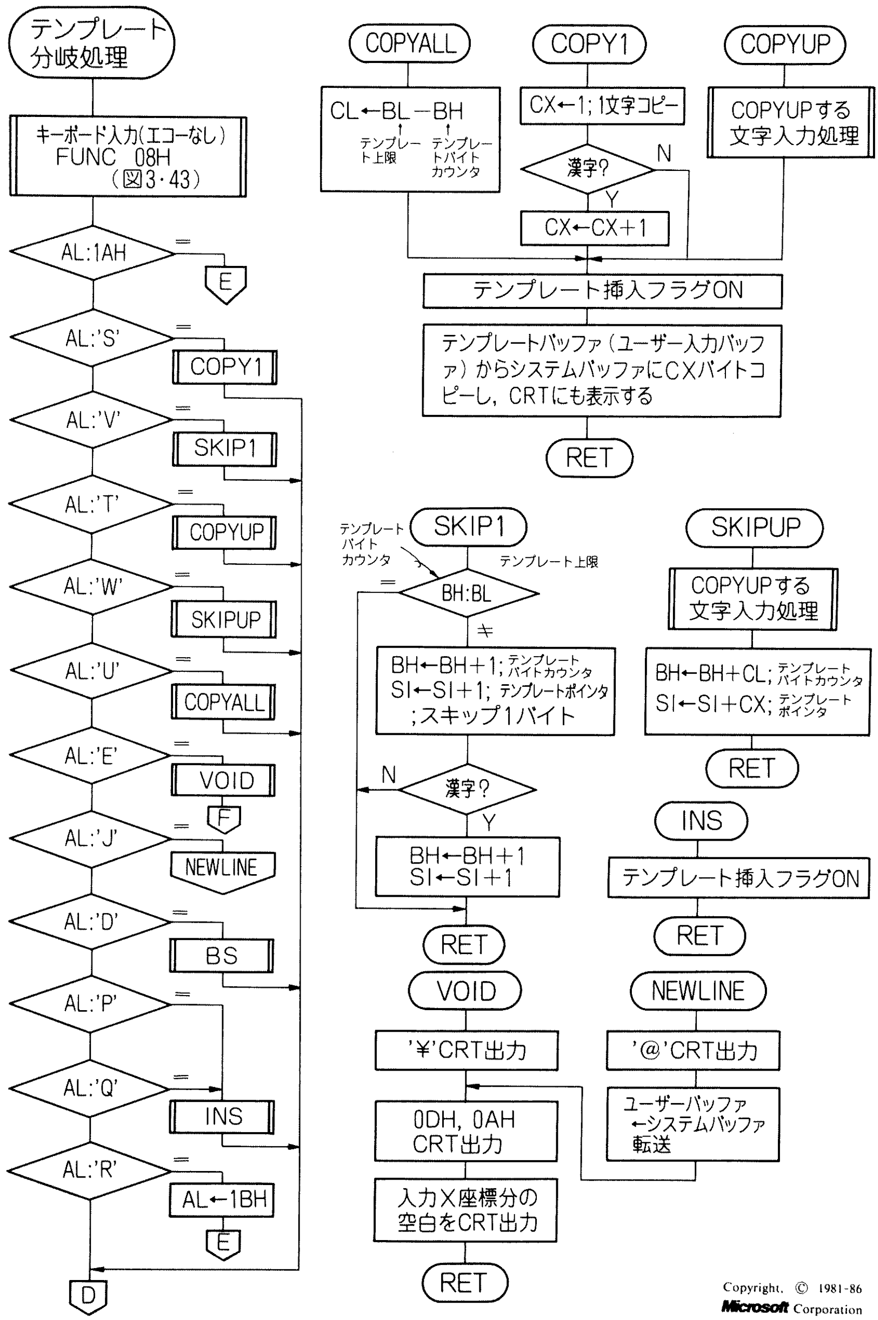


図 3・48(b) ファンクション 0AH バッファードキーボード入力解析フローチャート

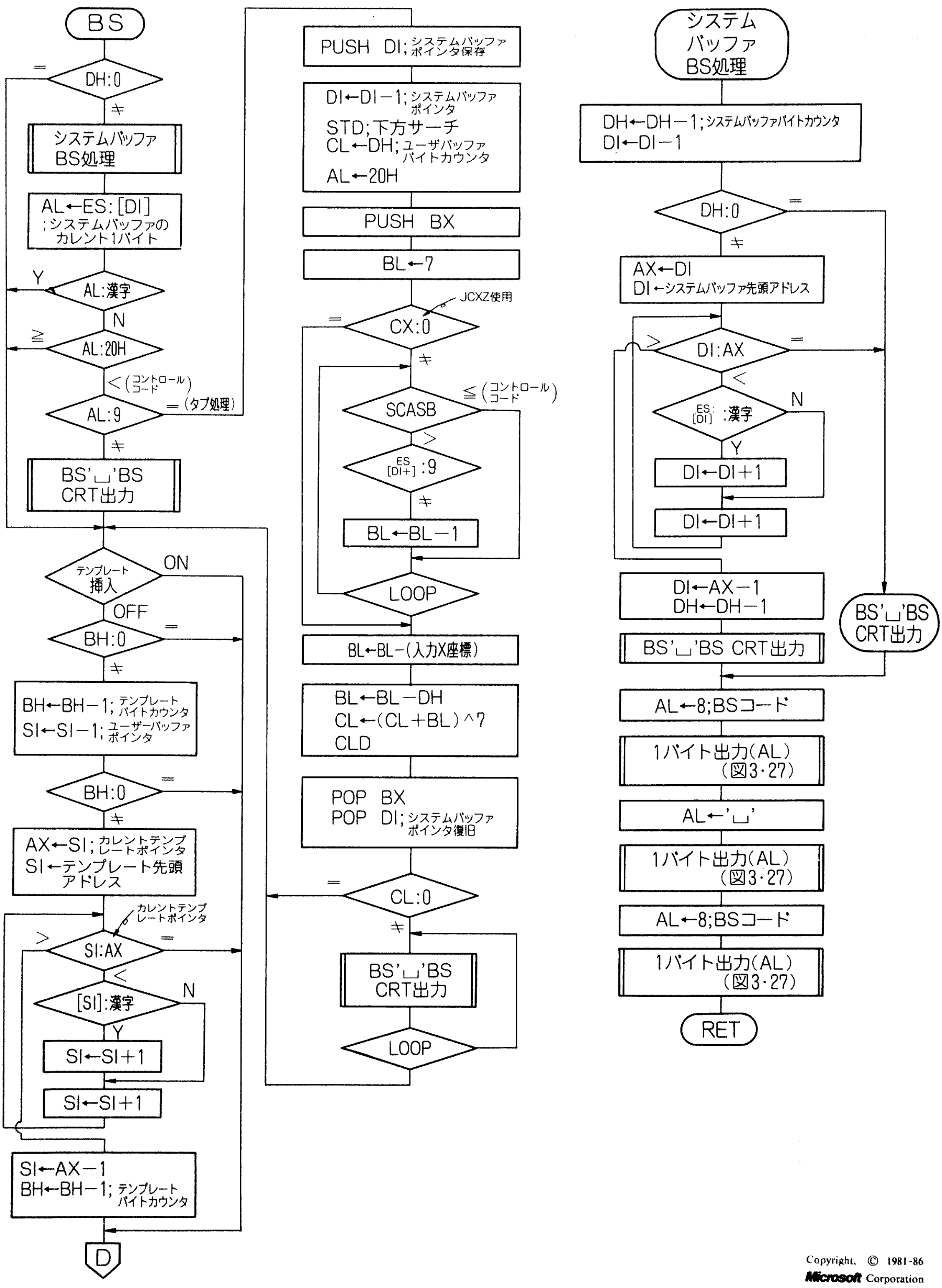


図 3・48(c) ファンクション 0AH バッファードキーボード入力解析フローチャート

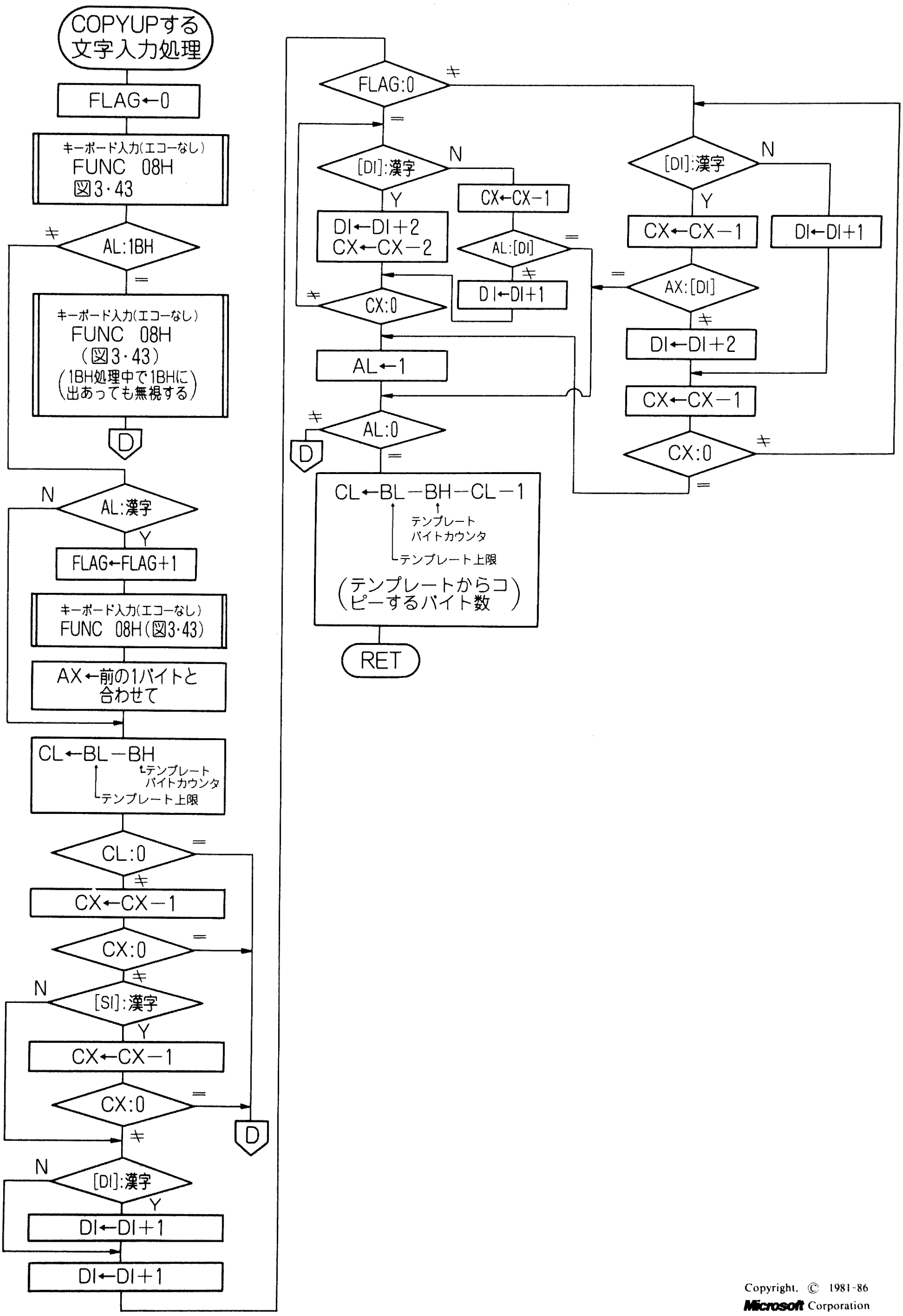


図3・48(d) ファンクション0AHバッファードキーボード入力解析フローチャート

漢字の判定について

MS-DOS のシフト JIS 漢字コードは 1 漢字 2 バイトで構成されますが，その 1 バイト目が必ず 81H ～9FH，E0H～FCH で始まるようになっています．そうなるように JIS の漢字コードが"シフト"されているのです．

こうすると，その 1 バイトが漢字エリアにあればつぎの 1 バイトと合わせて 1 つの漢字となることがわかります．フローチャートの中で漢字の判定をしているのは，このように"漢字の 1 バイト目かどうか"を判定しているのです．実際には漢字の判定は図 3・49 のように行っており，出力条件として

ZF ： 0 …… (NZ) 漢字である
　　　1 …… (Z) 漢字でない

となっています（ZF：ゼロフラグ）．

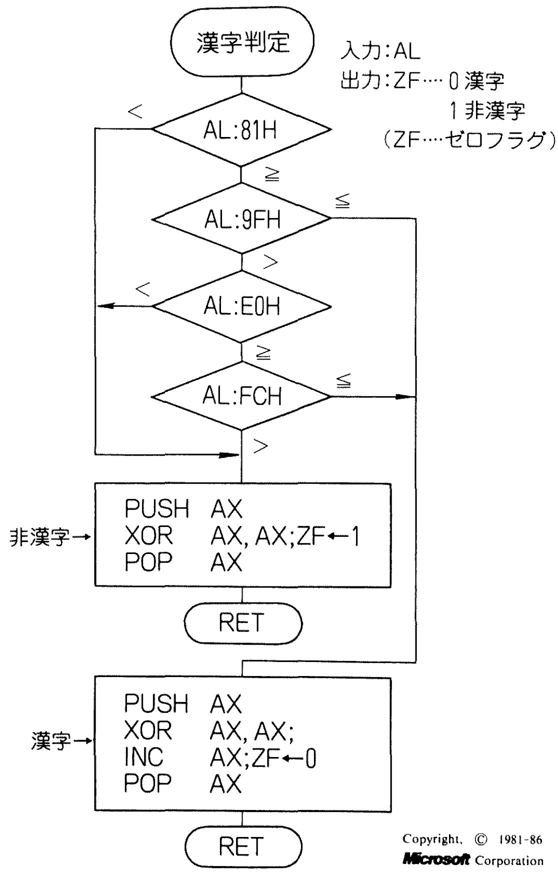


図 3・49　漢字判定フローチャート

ファンクション0BH
キーボードステータスチェック

"^C, ^S, ^P, ^N チェックルーチン（図 **3・36** 参照）"を CALL して，入力があれば AL に FFH を，なければ 00H をセットしているだけです．キーウェイトはありません．

図 3・50 に入出力条件，**図 3・51** に解析フローチャートを示します．

ファンクション番号	0BH
名　　称	キーボードステータス・チェック
入力条件	な　　し
出力条件	AL { 00H ……キー入力なし FFH ……キー入力あり

図 3・50　ファンクション 0BH　キーボードステータスチェック　入出力条件

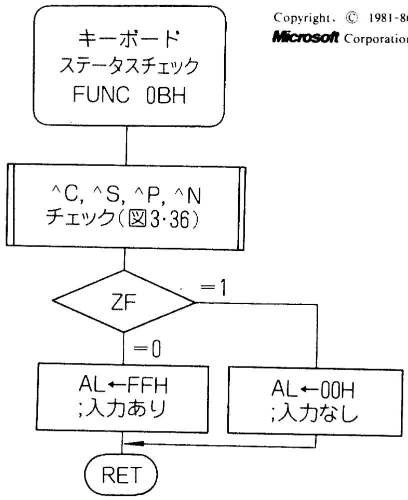


図 3・51　ファンクション 0BH　キーボードステータスチェック解析フローチャート

ファンクションキーと表示

MS-DOS を立ち上げると，ファンクションキーにはテンプレートコマンドがセットされていますが，画面には"C1　CU　CA"などと表示されています．ところがファンクションキーを押しても，C1　CU などとは表示されず，テンプレートコマンドが実行されます．これはファンクションキーにコントロールコードなどをセットした場合，その意味を表示するためのつぎのような機能を持たせてあるからです．

すなわち，ファンクションキーにセットするデータの最初の 1 バイトが FEH であれば，それに続く 5 バイトを無視するのです（しかし画面には表示します）．

そこでたとえばファンクションキー 1 には

08H　FEH　20H　43H　31H　20H　20H　1BH　53H

- 08H：データ数（8 バイト）
- FEH：特殊コード（以下の 5 バイトは無視せよ）
- 43H 31H：C1
- 1BH 53H：ESC　S

というコードがセットされていて，FEH 以下の 5 バイト'␣C1␣␣'が無視され（キー表示はされる），1BH　53H だけが実行されることになります．

ファンクション0CH
バッファを空にして キーボード入力

I/O リクエストコールの"タイプアヘッドバッファクリア"を実行したあと，AL レジスタにセットされていたキー入力ファンクション番号を AH レジスタに移し，INT 21H のファンクション 0CH 用の再エントリ（**図 3•16** 参照）へ JMP します．ただし，キー入力ファンクションとして許されているのはファンクション 1，6，7，8，0AH だけです．それ以外のコードを AL レジスタにセットすると，タイプアヘッドバッファをクリアするだけで終了します．

図 3・52 に入出力条件，**図 3・53** に解析フローチャートを示します．

ファンクション番号	0CH
名　　称	バッファを空にしてキーボード入力
入力条件	AL …バッファを空にしたあと実行されるファンクション番号（1，6，7，8，0AH） 各入出力条件は各ファンクション参照
出力条件	○上記以外の番号でもバッファクリアだけは行われる ○上記以外の番号のとき AL ← 0

図 3・52 ファンクション **0CH** バッファを空にしてのキーボード入力入出力条件

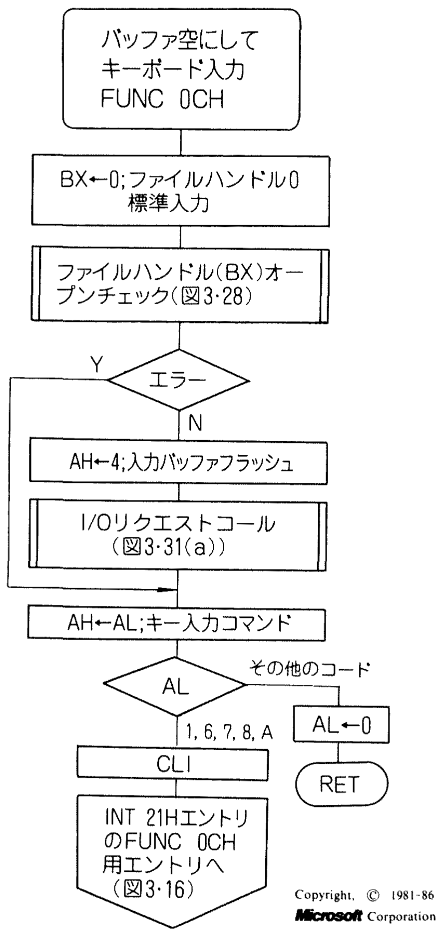


図 3・53 ファンクション **0CH** バッファを空にしてのキーボード入力解析フローチャート

ファンクション0DH
ディスクのリセット

全ドライブバッファをフラッシュし，特定のドライブの使用フラグが立っていたらクリアします．すなわち，特定のドライブ用として使用していたらバッファの（+4）にそのドライブ番号が書かれているので，FFH（使用ドライブなし）を書き込みます．また，できるだけ未使用のバッファをリンクの前へ持ってくるようにバッファの再編成をします．

図3・54に解析フローチャートを示します．

ディスクバッファ・フラッシュ

ファンクション0DH(ディスクのリセット)，"ALで指定されるドライブのバッファフラッシュ"でも使用されています．ディスク関係の処理ルーチンではディスクに書き込む必要があると，バッファのオフセット（+5）に1を書き込み，バッファに書き込むべきデータが入っていることを印しておきます．このとき書き込むべきドライブ番号をバッファの（+4）へ，セクタ番号を（+8，9）へ記録しておきます．このようなバッファを"未フラッシュのバッファ"といいます．そして未フラッシュのバッファでいっぱいになったときや処理の要所要所でこの"ディスクバッファ・フラッシュルーチンをCALLして，まとめて書き込みます．

ALレジスタで指定したドライブだけフラッシュすることができます．ドライブ番号の数え方はA→0，B→1，C→2，……となっています．FFHを指定すると全ドライブのバッファとなります．バッファフラッシュしたあとは未フラッシュフラグ（+5）はクリアしますが，ドライブ番号（+4）とセクタ番号（+8，9）はクリアせずに保存します．これはディスク読み込みのとき，はじめにバッファの中を探索し，読み込むべきドライブ・セクタのデータが入ったバッファを見つけるときに使おうというわけです．バッファの中に読み込むべきドライブ・セクタのデータがないとき，はじめてドライブをアクセスするのです．したがってバッファの数が多いとバッファの中でヒットする確率が高くなり，ディスクをアクセスする回数を減らし速度UPとなります．これを"バッファリング効果"といいます．

図3・55にディスクバッファフラッシュの解析フローチャートを示します．

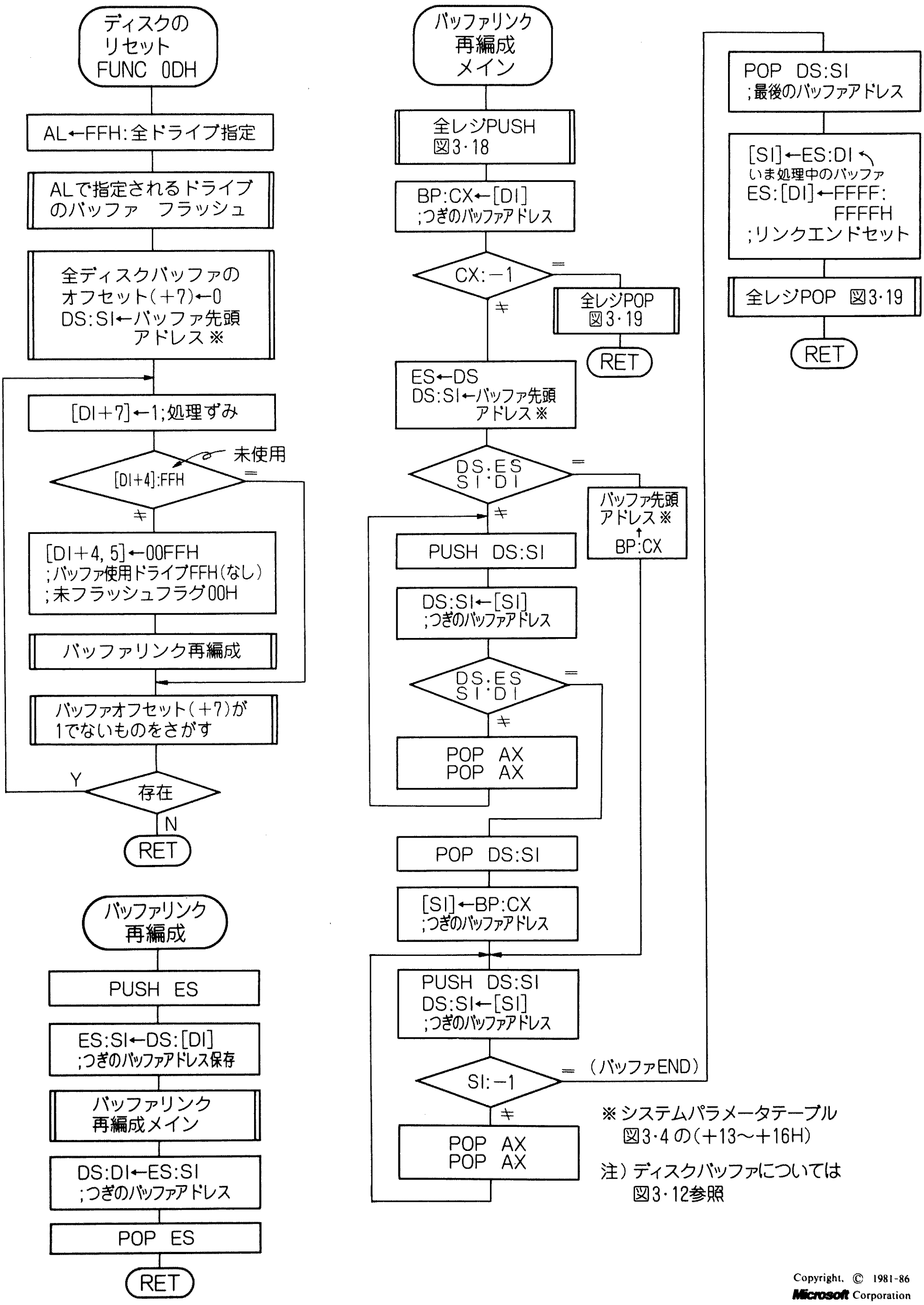


図3・54 ファンクション0DH ディスクのリセット解析フローチャート

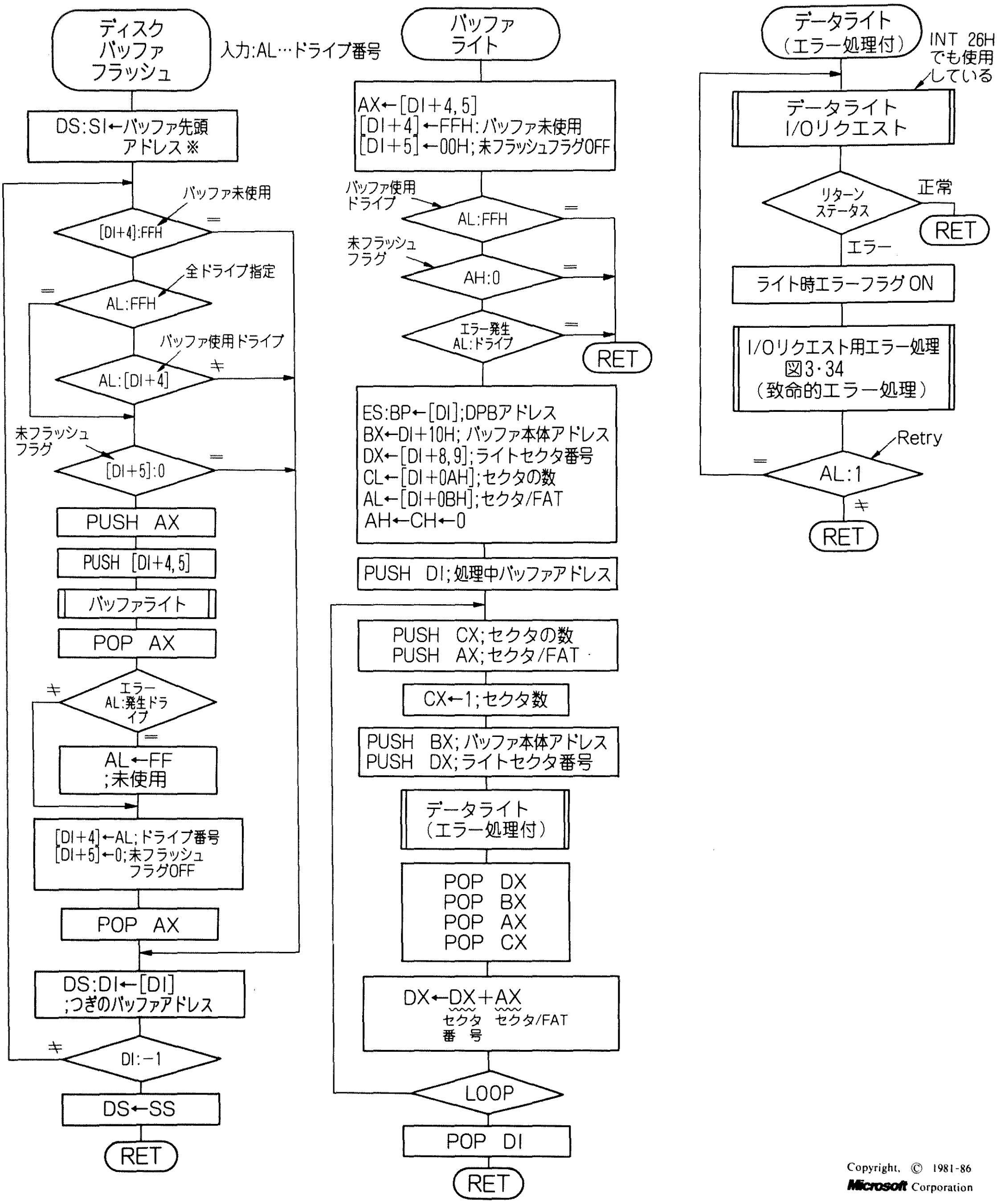


図 **3・55** ディスクバッファフラッシュ解析フローチャート

ファンクション0EH
ドライブの選択

DL で指定されたドライブ番号を接続されているドライブ数(これは DOS のイニシャライズ時にセンスしてシステムのワークエリアに保存されている．図 3•4 参照) と比較してオーバーしていればそのままリターンし，範囲内であればカレントドライブを記憶しているシステムワークエリアに DL を書き込みます．

いずれの場合にも AL レジスタに接続ドライブ数がリターンされます．

図 3・56 に入出力条件，図 3・57 に解析フローチャートを示します．

ファンクション番号	0EH
名　　称	ドライブの選択
入力条件	DL …カレントドライブにしたいドライブ番号
出力条件	AL …接続ドライブ数

図 3・56 ファンクション 0EH ドライブの選択 入出力条件

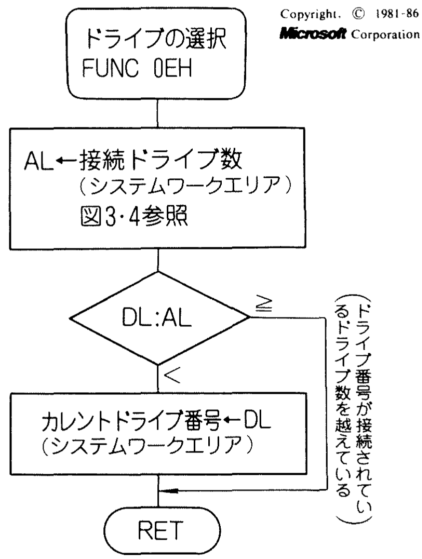


図 3・57 ファンクション 0EH ドライブの選択 解析フローチャート

ファンクション0FH

FCBによるファイルオープン

まずDS：DXで指定されたFCBのファイル名がカレントディレクトリに存在するかチェックします(この処理の中で拡張FCBのチェックもする)．存在しなければALにFFHを入れてリターンします(エラー)．存在すれば(これ以降の処理は"FCBによるファイル作成，ファンクション16H"や"パス名によるファイル作成，ファンクション3DH"からも使用されている)FCB＋18Hのビット7を見てキャラクタ型かブロック型か判断し，処理を分けています(bit7＝1のときキャラクタ型(**図3・7**参照))．

ファンクション0FHはすでにディレクトリに存在するファイルのオープンです．ディレクトリに存在しないときはエラーになります．ディレクトリに存在しないファイルをオープンして使いたいときには，ファイルのクリエイト（ファンクション16H）を使用します．

ディレクトリにファイルが存在し，オープンされたあとにはFCBには以下のような値がセンスされてリターンします．

FCB

+0	…ブロック型はドライブ番号（1→A，2→B…） キャラクタ型は0
+0C，DH	…カレントブロック　0
+0E，FH	…レコードサイズ　80H（＝128バイト）
+10～13H	…ファイルサイズ
+14，15H	…最終書き込み日付
+16，17H	…最終書き込み時刻
+18H	…ブロック型はドライブ名 キャラクタ型はデバイス属性を8ビットに圧縮したもの
+19，1AH	…ブロック型は最初のクラスタ キャラクタ型は＋19H～1CHにデバイスヘッダアドレス
+1B，1CH	…ブロック型のみ　0
+1DH	…ブロック型のみ最初のクラスタ番号
+1E，1FH	…ブロック型のみカレントディレクトリのクラスタ番号

図3・58に入出力条件，**図3・59**に解析フローチャートを示します．

ファンクション番号	0FH
名　　称	FCB によるファイルオープン
入 力 条 件	DS：DX…FCB アドレス ↓ ドライブ番号 / ファイル名 / 0 クリアのこと
出 力 条 件	AL …00H オープン成功 FFH オープン失敗 （ディレクトリエントリが存在しない）

図 3・58　ファイル 0FH　FCB によるファイルオープン　入出力条件

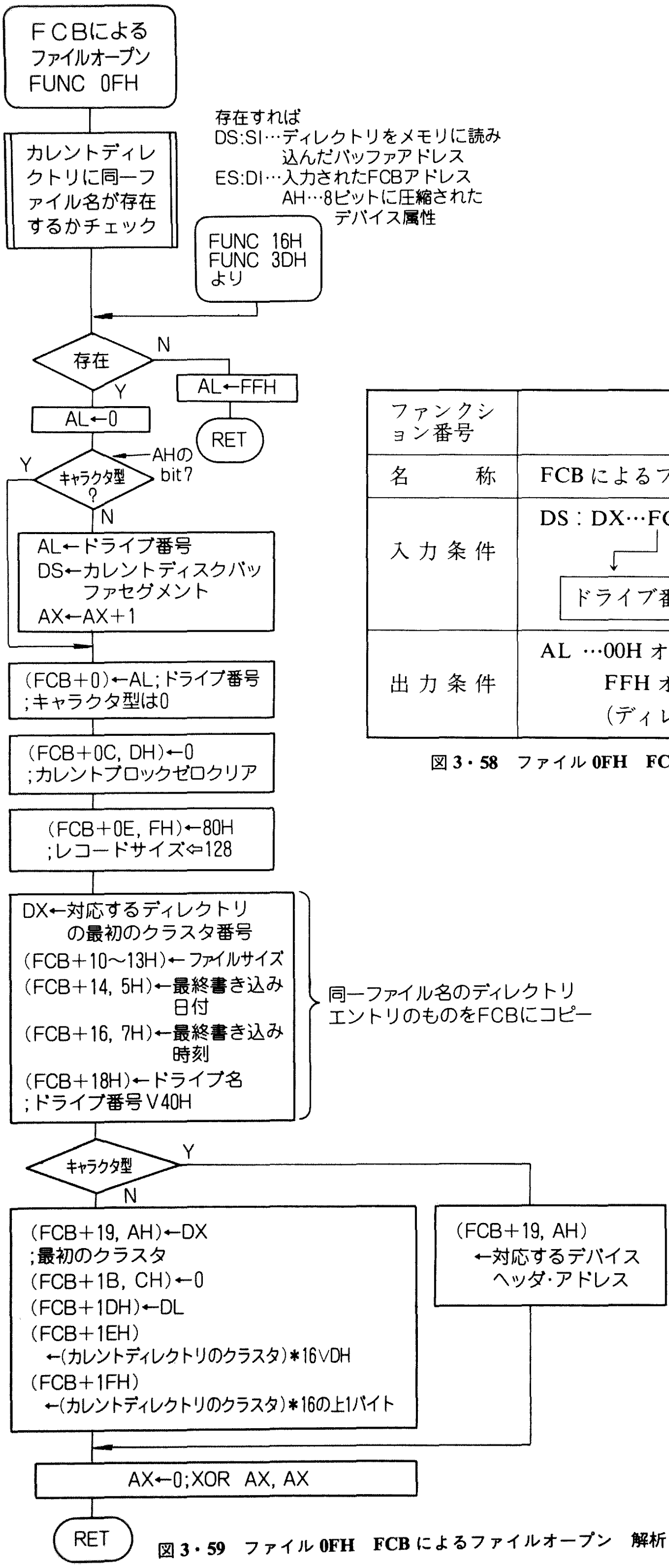


図 3・59　ファイル 0FH　FCB によるファイルオープン　解析フローチャート

ファンクション10H
FCBによるファイルクローズ

FCB (DS：DX でアドレス指定) と同じファイル名のディレクトリエントリをさがしだしてきて，メモリ上のカレントディスクバッファにそのディレクトリエントリが含まれる1セクタを読み出します．それからFCB上の最初のクラスタ，ファイルサイズ，最終変更日付，最終変更時刻をバッファに読み込んだディレクトリエントリへコピーします．カレントディスクバッファの未フラッシュフラグをONにして，FCBに指定するドライブだけバッファフラッシュ（**図3･55**参照）して処理終了です．

したがって，日付･時刻，最初のクラスタ，ファイルサイズなどはクローズされるまでFCB上に保存されていることになります．

入出力条件と解析フローチャートを**図3・60**，**図3・61**に示します．

メディアチェック&DPB更新

I/Oリクエストコール"メディアチェック（コマンド番号1）"を実行し，メディア交換されていなければそのままリターンし，交換されているか，交換不明のときはBPB作成（I/Oリクエストコールコマンド番号2）を実行したのち，BPBからDPBを作成します．

図3・62に解析フローチャートを示します．

ファンクション番号	10H
名　　称	FCB によるファイルクローズ
入力条件	DS : DX…FCB アドレス
出力条件	AL … 0 ディレクトリエントリ存在し，クローズ成功 FFH ディレクトリエントリが存在しない．

図 3・60 FCB によるファイルクローズ　入出力条件

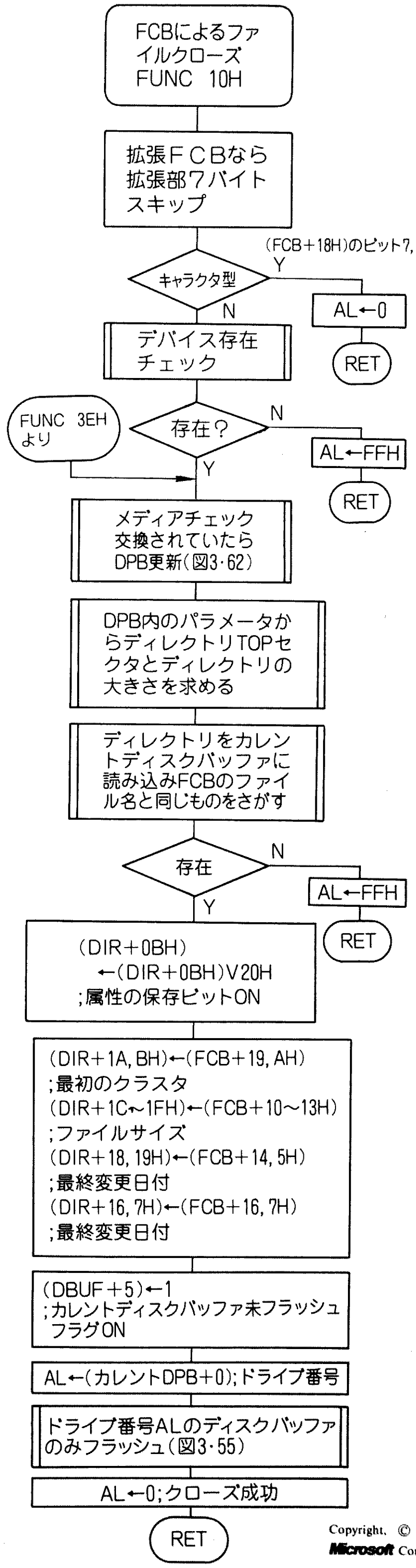


**図 3・61 FCB によるファイルクローズ
解析フローチャート**

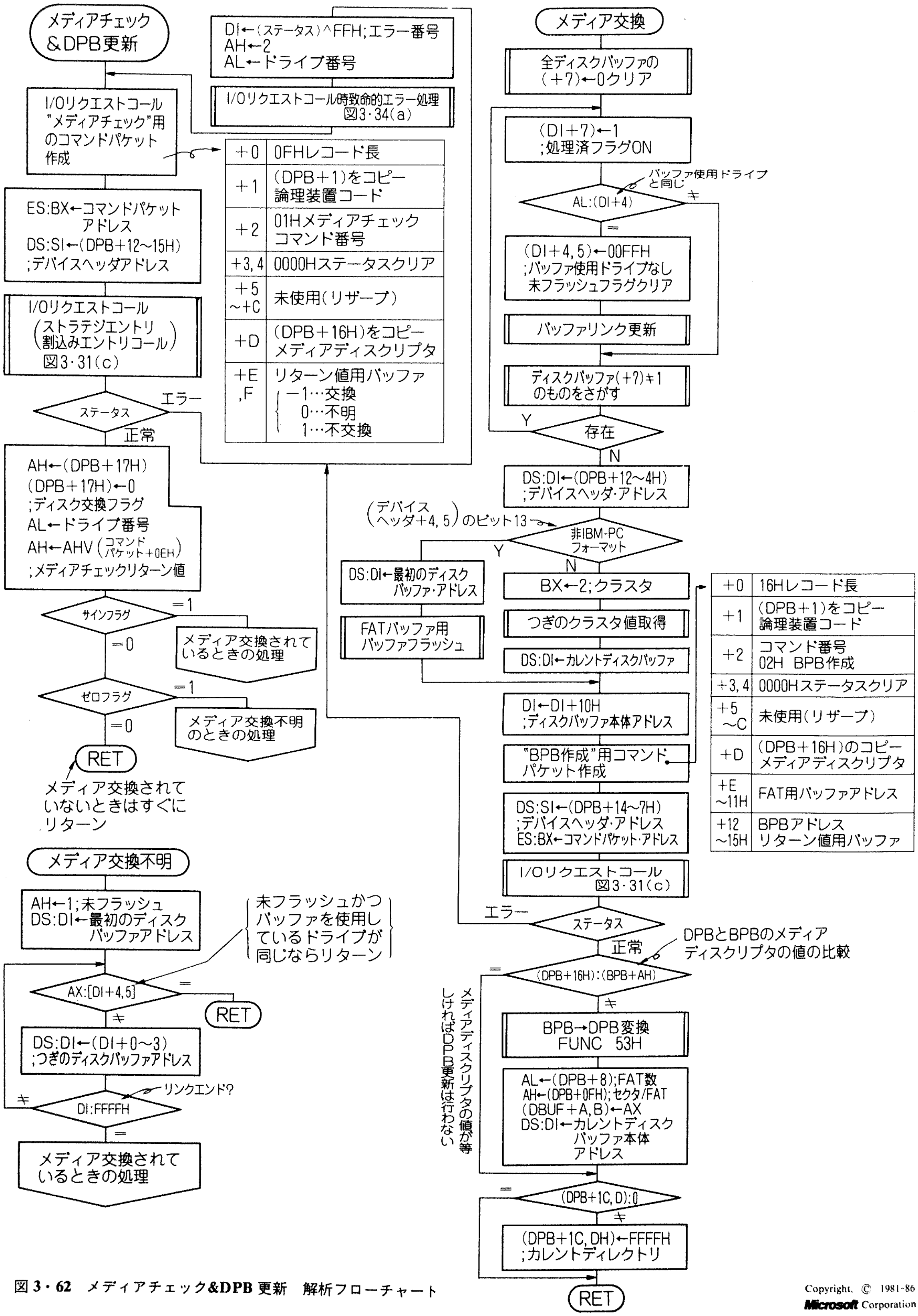


図 3・62 メディアチェック&DPB 更新 解析フローチャート

ファンクション11H
FCBによる 最初のディレクトリ検索

ファンクション12H
FCBによる つぎのディレクトリ検索

カレントディレクトリの中からFCBと同じファイル名のものをさがします．ファンクション11Hのほうはカレントディレクトリ先頭からさがしていくのに対して，ファンクション12Hは前にさがした続きから後をさがします．ファンクション12Hは，FCB内に記録されている"前に何番目のエントリまで検索したか（FCB+10，11H）"の情報をもとに検索するため，最初はかならずファンクション11Hで検索しておかねばなりません．2回目からはファンクション12HでOKです．検索のために用意したFCBのファイル名の先頭がE5Hのとき，誤っていったん削除されたファイルを検索してしまう可能性があります．そのため検索ファイル名がE5Hのときは05Hに書き直してファイル名比較をしています．

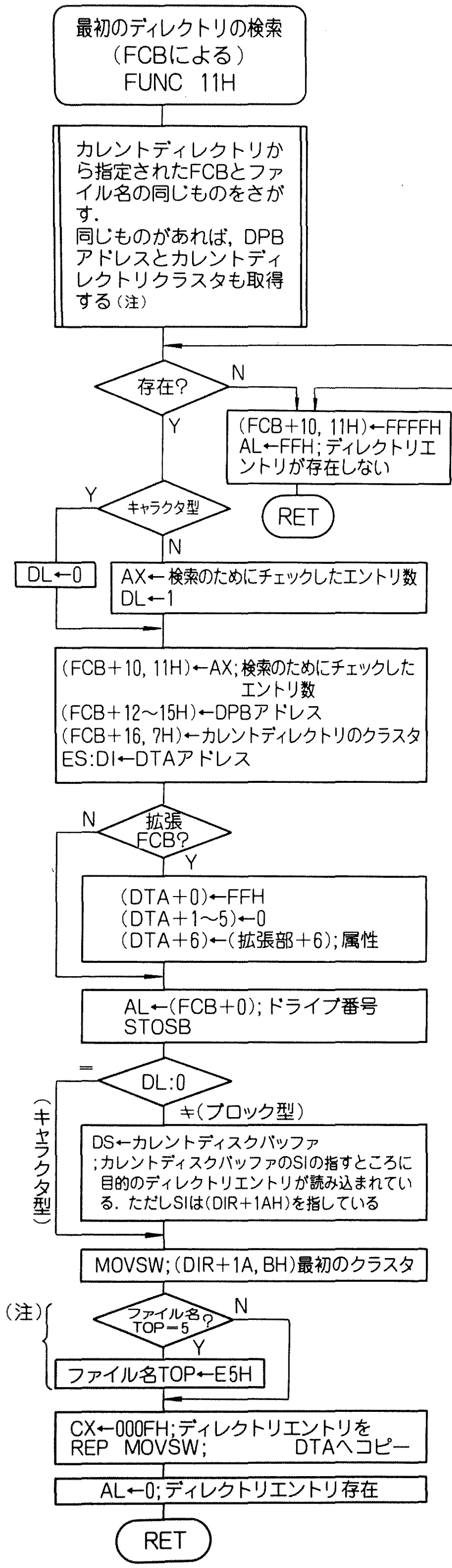
検索のためのFCBは

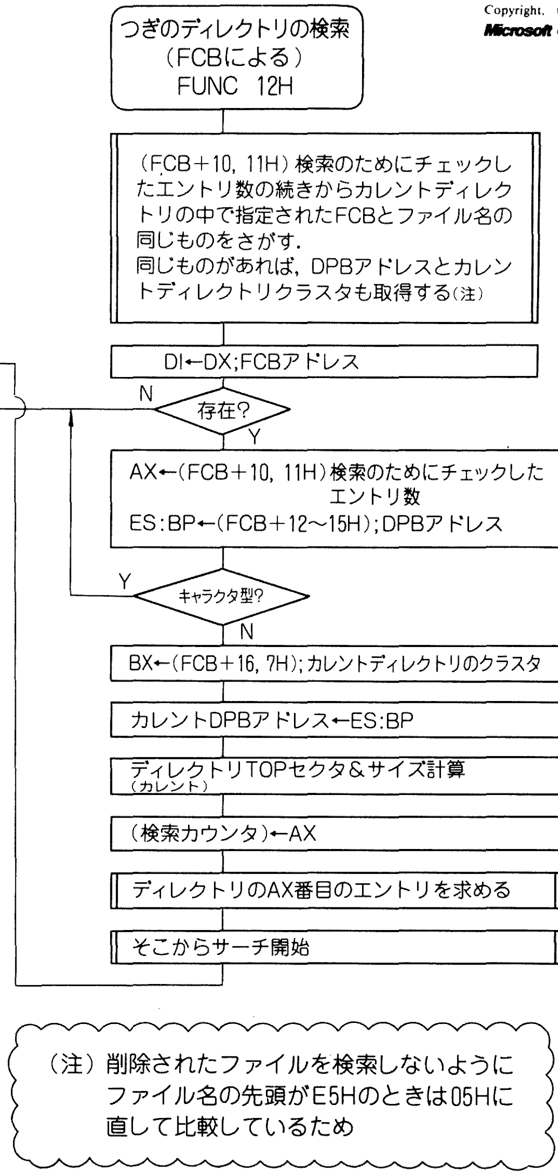
FCB
+0 … ドライブ番号
+1〜8 … ファイル名
+9〜0BH … 拡張子
⋮
+10，11H … 検索のためにチェックしたディレクトリエントリ数
+12〜13H … DPBアドレス
+16，17H … カレントディレクトリのクラスタ
⋮

として使われます．ファイルオープンなどでの使われかたとはすこし違います．このように使うのはファイル検索のときだけです．

したがって検索ファンクションを実行したあとで（FCB+10，11H）を見れば，そのときのカレントディレクトリの先頭から何番目のディレクトリエントリかということもわかります．

図3・63に入出力条件と解析フローチャートを示します．





ファンクション番号	11H, 12H
名称	FCBによる最初のディレクトリ検索 FCBによるつぎのディレクトリ検索
入力条件	DS:DX←FCBアドレス
出力条件	AL…00H ディレクトリエントリ存在 FFH 〃 存在しない

図3・63 ファンクション **11H** 最初のディレクトリ検索
ファンクション **12H** つぎのディレクトリ検索
の入出力条件と解析フローチャート

ファンクション13H

FCBによるファイル削除

拡張 FCB に対する処理は"デバイス存在チェック"で行われます．ファイル名の検索は FCB で指定したドライブのカレントディレクトリをカレントディスクバッファへ読み込んで，FCB のファイル名と比較して行います．

リターン値 AL に FFH が返される場合は

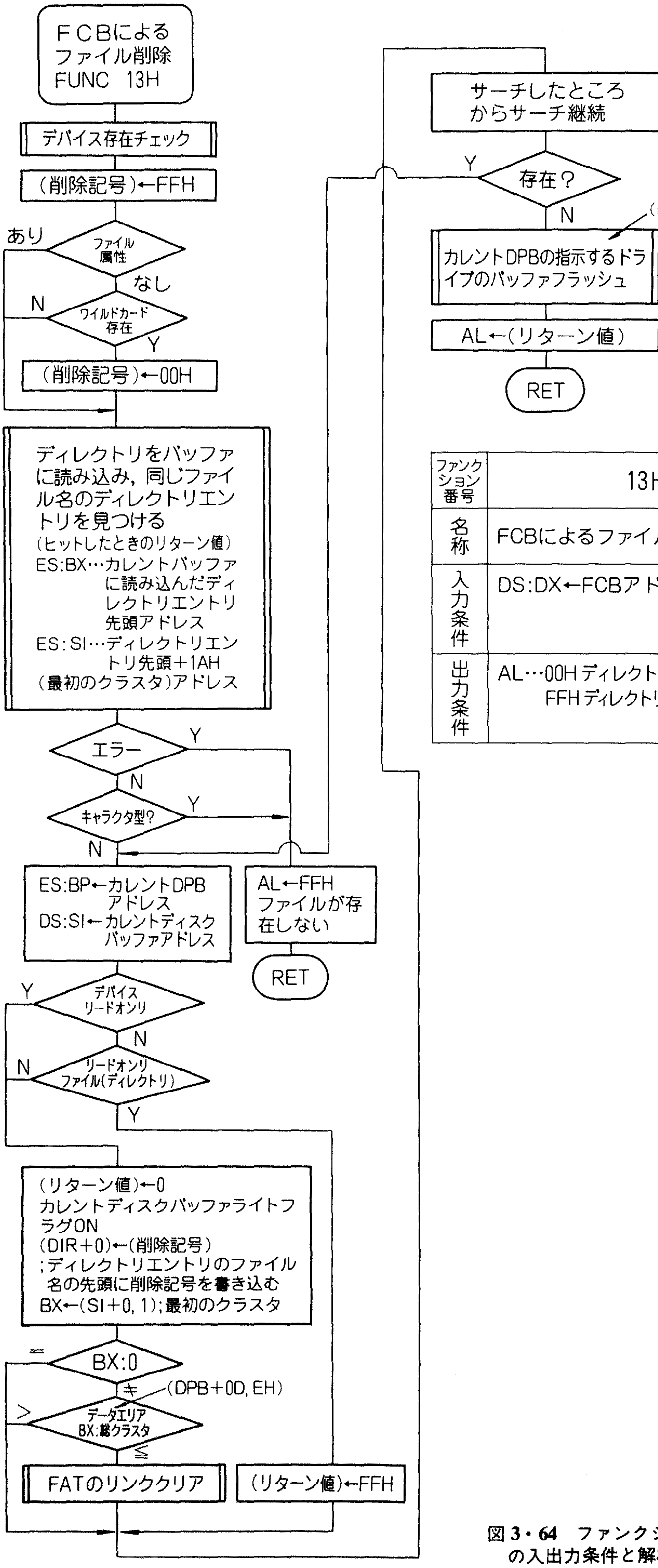
① キャラクタ型のファイルを削除しようとしたとき

② ディレクトリに FCB で指定されたファイル名が存在しないとき

③ FCB のファイル名にワイルドカードを使ったとき，ワイルドカードと一致するファイルに 1 つでもリードオンリファイルが存在したとき

です．一致するファイル名が見つかったときはディレクトリエントリの先頭に削除記号を書き込むとともに，そのファイルが使用している FAT のリンクをクリアします．

図 3・64 に入出力条件と解析フローチャートを示します．



ファンクション番号	13H
名称	FCBによるファイル削除
入力条件	DS:DX←FCBアドレス
出力条件	AL…00H ディレクトリエントリ存在 FFH ディレクトリエントリ存在しない

図3・64　ファンクション13H　FCBによるファイル削除の入出力条件と解析フローチャート

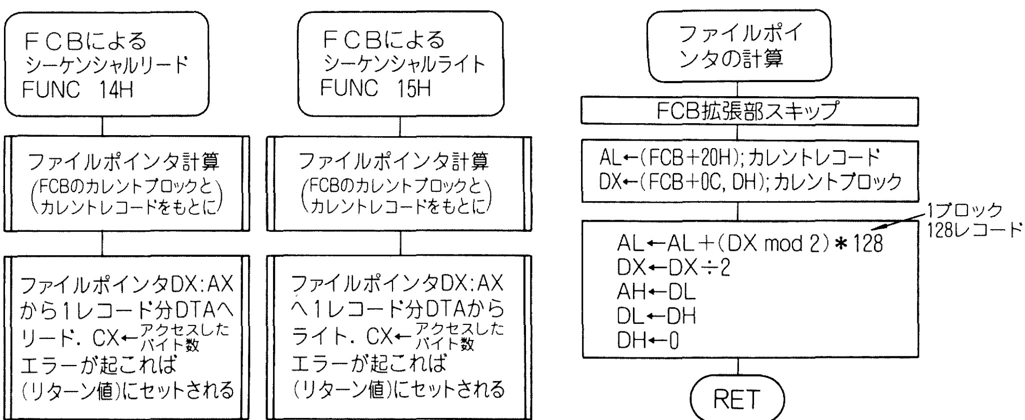
ファンクション14H
FCBによる シーケンシャルリード

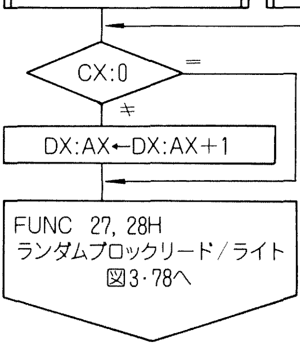
ファンクション15H
FCBによる シーケンシャルライト

シーケンシャルなファイルアクセスは，カレントブロック(FCB+0C, DH)とカレントレコード(FCB+20H)を使って行われますが，これら2つの値からファイルポインタ（ファイル先頭から何レコード目かということ）を計算することによって行います．アクセス終了後は逆にファイルポインタをカレントブロックとカレントレコードに換算して，FCBにセットしています．また，1回のアクセスするバイト数はレコードサイズによって決定できます．

出力条件に返される値は"ファイルポインタDX：AXから1レコード分DTAへリード/ライト"ルーチンの中でセットされます．

図3・65に入出力条件と解析フローチャートを示します．





ファンクション番号	14H	15H
名称	FCBによるシーケンシャルリード	FCBによるシーケンシャルライト
入力条件	DS:DX…FCBアドレス	
出力条件	AL…0…正常終了 1…アクセスしたレコードにデータが入っていない 2…DTAの大きさが1レコードの大きさより小さい 3…ファイルの終わりまで読み出した．(バッファの残りは0でうめる)	AL…0…正常終了 1…Disk Full 2…DTAの大きさが1レコードより小さい

図3・65 ファンクション**14,15H** **FCB**によるシーケンシャルリード/ライト入出力条件と解析フローチャート

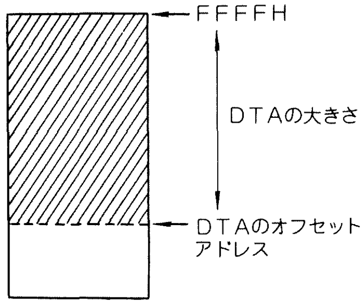


図3・66 FCB の大きさ

FCB のカレントブロックとカレントレコードから，ファイルポインタはつぎのように行っています.

まず，FCB の拡張部があればスキップします.

```
AL  ←  (FCB+20H)      ;カレントレコード
DX  ←  (FCB+0C, DH)   ;カレントブロック
SHL AL, 1             ; ┃ DXは右へ1ビットシフト（1/2倍）し
SHR DX, 1             ; ┃ 最下位ビットはALの最上位ビットと置きかえる
RCR AL, 1
AH  ←  DL
DL  ←  DH
DH  ←  00H
```

これで DX：AX にファイルポインタ（レコード単位）ができあがります.

● **DTA**[注] **の大きさ**

入出力条件の中で”DTA の大きさ”といっているのは，DTA の先頭アドレスのオフセット値から上のセグメント境界（オフセット FFFFH）までのバイト数のことです（図 3・66 参照).

たとえば DTA のオフセット値が 80H ならば

FFFFH-80H＝FF7FH

となります. すなわち，レコードサイズは FF7FH 以下でなければならないことになります. もちろん，そのためにプログラムエリアをこわしてはいけません. DTA のうしろにあるプログラムエリアやデータエリアを不注意なレコードサイズ指定でこわしてしまわないためには，DTA のオフセット・セグメントを DTA の上限が FFFFH になるように選ぶようにすれば大丈夫です. そうすれば DTA の上限 FFFFH を越えるアクセスをしたときには，エラーコード 2 によって処理が中断され，誤ってコード（データ）を破壊することはありません.

シーケンシャルリードの出力条件のエラーコード 1 に，”アクセスしたレコードにデータが入っていない”というのがありますが，実際には”ファイルポインタ（カレントブロック：カレントレコード）がファイルの外を指している”ということです.

注） DTA とは’Data Transfer Area’の略で，システムとあるまとまったデータをやりとりするときに使うバッファです. なにも指定しない状態では PSP（図 3・8 参照）の中にあります. DTA アドレスは PSP 以外にもセットできます（ファンクション 1AH，図 3・70 参照).

ファンクション16H

FCBによる ファイル作成（クリエイト）

ディレクトリに新しくディレクトリエントリを作成するファンクションです．指定したファイル名がすでに存在するときは，そのファイルがボリュームラベル，システムファイル，不可視属性がつけられていないときにファイルの内容をクリアして，ファイルサイズなどを設定して新規ファイルとします．キャラクタ型だった場合はつねに存在とみなされ，日付・時刻などの設定は行われません．

指定したファイル名と同じものがないときは，新しくディレクトリエントリを確保して新規ファイルとします．新規ファイル作成の処理が終われば"FCBによるファイルオープン（ファンクション0FH，**図3・59**参照）"へ飛びます．そこで具体的にFAT上に最初のクラスタが割り当てられるのです．

リターン値としてALにFFHが返されるとエントリとなりますが，この場合として

① FCBで指定されたファイルがキャラクタ型のとき対応するデバイスがない

② FCBで指定されたファイル名の中にワイルドカードがある

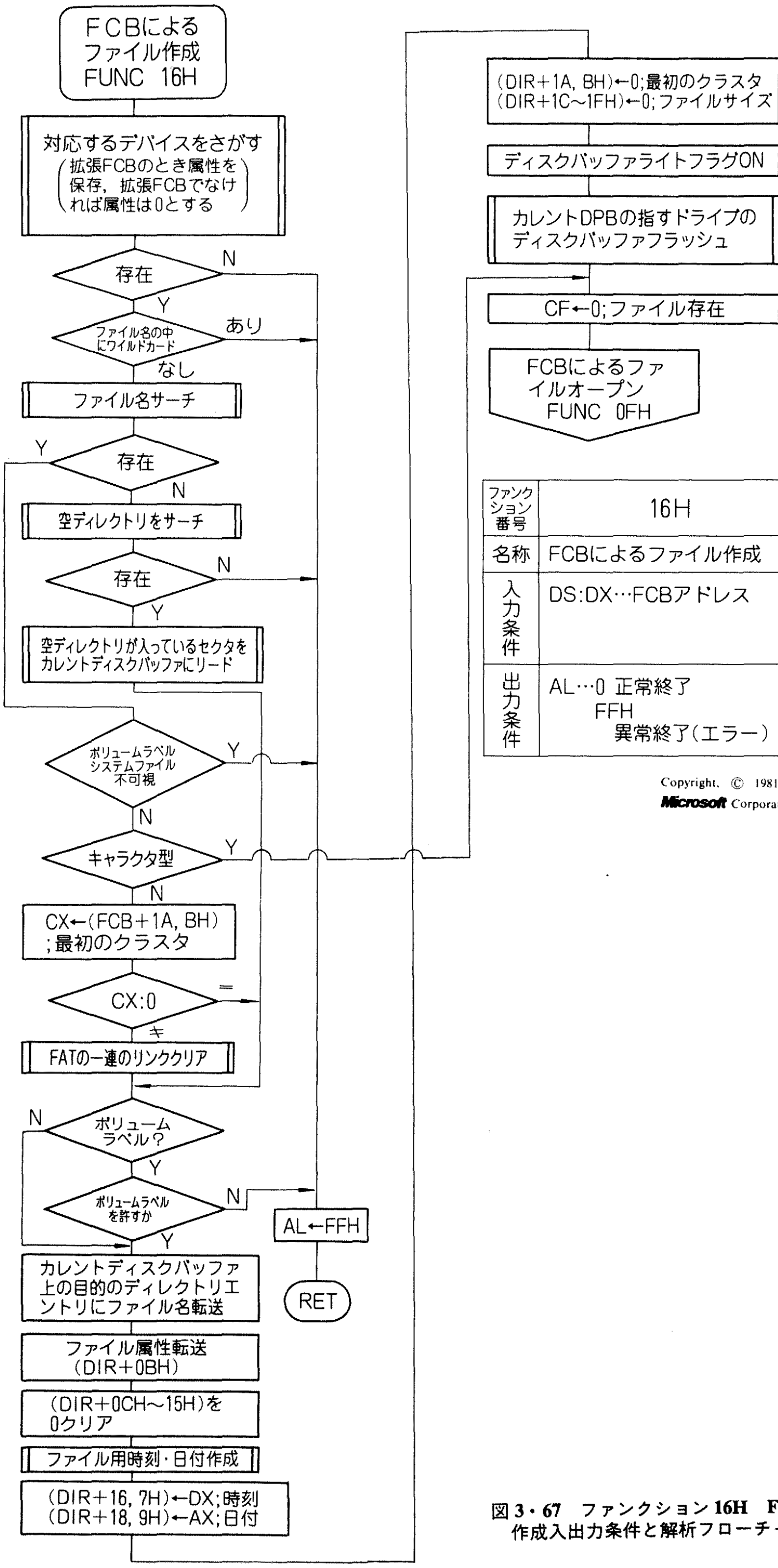
③ FCBで指定されたファイル名がディレクトリ中にないとき新しくディレクトリエントリを作成するスペースがない

④ FCBで指定されたファイル名がディレクトリ中にあるが，ボリュームラベル・システムファイル・不可視属性がついている

⑤ 拡張FCBでボリュームラベルを指定したとき，システムワークエリア内のボリュームラベルを許さないフラグが立っている

ということが考えられます．通常システムワークエリア内のボリュームラベルを許さないフラグは降りていますので，拡張FCBで属性にボリュームラベルを指定すれば作成することができます．

図3・67に入出力条件と解析フローチャートを示します．

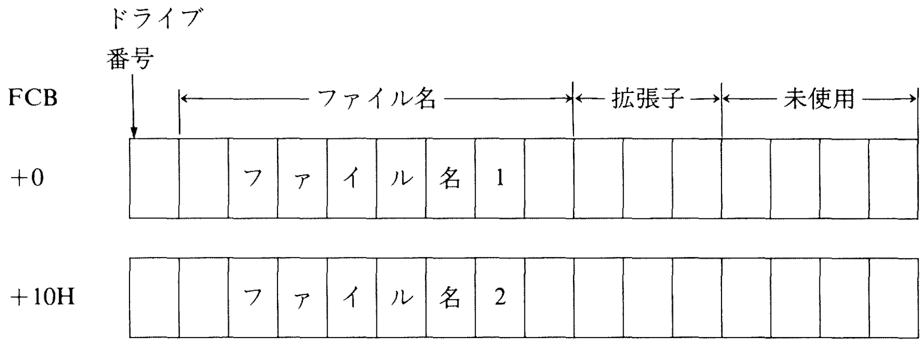


ファンクション番号	16H
名称	FCBによるファイル作成
入力条件	DS:DX…FCBアドレス
出力条件	AL…0 正常終了 FFH 異常終了（エラー）

図3・67 ファンクション16H FCBによるファイル作成入出力条件と解析フローチャート

ファンクション17H
FCBによるリネーム

リネームの場合のFCBの使い方は，他のものとちょっと変わっています．



（注：四角1つは1バイト）

のように使って，ファイル名1をファイル名2に変更します(もちろん，拡張子も含む)．したがってデフォールトのFCBのファイル名がそのまま使えます．このとき

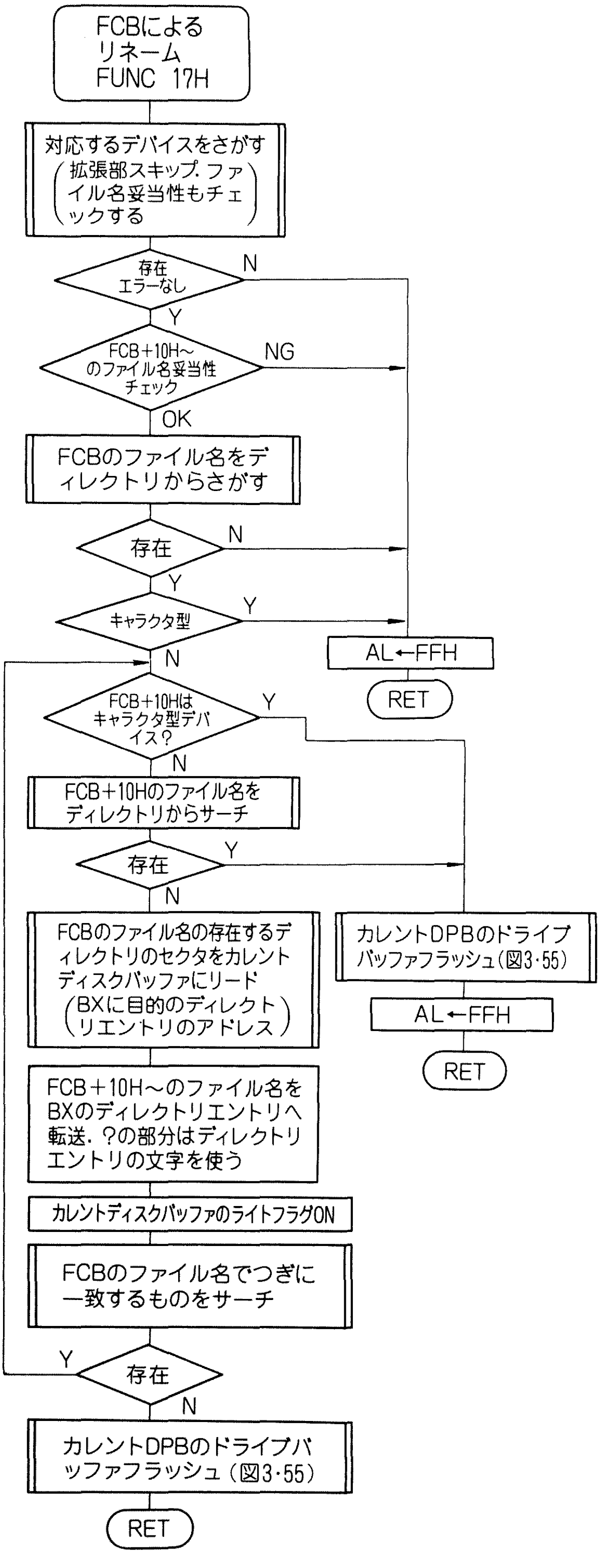
A>コマンド名　　ファイル名1 ファイル名2

とすると，PSPの5CH～がファイル名1，6CH～がファイル名2となります．

キャラクタ型のデバイス名はリネームの対象とはなりません．

ファイル名1のほうにワイルドカードを使うと，ファイル名と一致するディレクトリエントリをつぎつぎにリネームすることができます．ファイル名2にワイルドカードを使うと，?の部分はディレクトリエントリの対応する部分の文字が使われます(内部では＊は?に展開されている)．

図3・68に入出力条件と解析フローチャートを示します．



ファンクション番号	17H
名称	FCBによるリネーム
入力条件	DS:DX←FCBアドレス +0 ファイル名1 +10 ファイル名2
出力条件	AL…0 正常終了 FFH 異常終了

図3・68 ファンクション17H FCBによるリネームの入出力条件と解析フローチャート

ファンクション18H, 1DH, 1EH, 20H

ノーオペレーション

機能がなく，まったく何もしないファンクションです．ALに0を入れてリターンするだけです．

ファンクション19H

カレントドライブ番号リード

ファンクション番号	19H
名称	カレントドライブ番号リード
入力条件	なし
出力条件	AL…カレントドライブ番号

図3・69　ファンクション19Hの入出力条件

システム内のワークエリアからカレントドライブ番号をALレコードに読み出してリターンするだけです．

ファンクション1AH

DTAアドレスのセット

ファンクション番号	1AH
名称	DTAアドレスのセット
入力条件	DS:DX…DTAアドレス
出力条件	なし

図3・70　ファンクション1AHの入出力条件

システム内のワークエリアにDTAのアドレスをセットするだけです．DTAのアドレスの決め方はファンクション14H，15H(シーケンシャルリード/ライト)の項も参照してください．

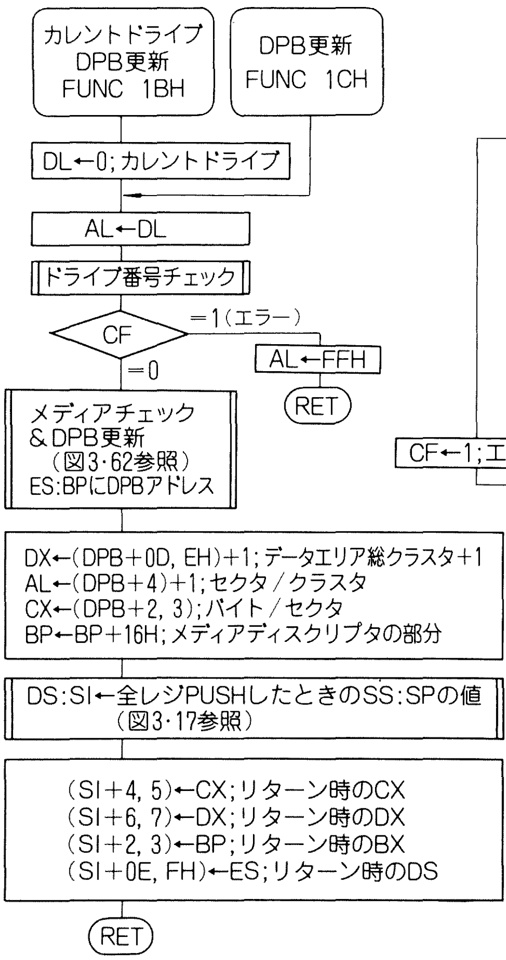
ファンクション1BH
カレントドライブDPB更新

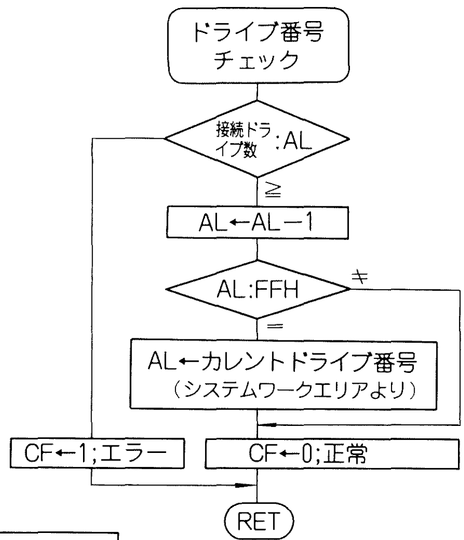
ファンクション1CH
DPB更新

ファンクション 1CH は DL レジスタの示すドライブに対して DPB 更新を行います.

ファンクション 1BH は DL=0(カレントドライブ)に対して DPB 更新をします.

入出力条件と解析フローチャートを**図 3・71** に示しますが, "メディアチェック&DPB 更新" の部分は**図 3・62** を参照してください.





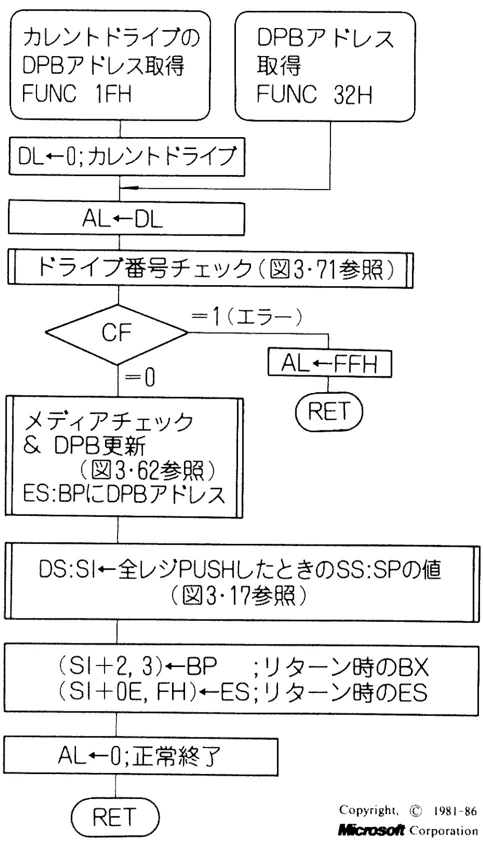
ファンクション番号	1BH	1CH
名称	DPB更新	
入力条件	なし	DL…ドライブ番号 (0…カレント)
出力条件	AL…FF 異常終了 その他セクタ/クラスタ CX…バイト/セクタ DX…データエリア総クラスタ+1 DS:BX…DPBのメディアディスクリプタの部分のアドレス(DPB+16H)	

図 3・71 ファンクション 1BH,1CH DPB 更新の入出力条件と解析フローチャート

ファンクション1DH, 1EH
ノーオペレーション

ファンクション 18H を参照してください.

ファンクション1FH
カレントドライブの DPBアドレス取得



ファンクション20H
ノーオペレーション

ファンクション 18H を参照してください.

ファンクション32H
DPBアドレスの取得

ファンクション 32H は DL で指定されたドライブの DPB アドレスを取得し，ファンクション 1FH は DL=0 (カレントドライブ) としたときの DPB アドレスを取得します．処理内容としてはファンクション 1BH, 1CH とほとんど同じです．ただファンクション 1FH, 32H のほうは，ストレートに DS : BX に DPB アドレスを返すだけです．

図 3・72 に入出力条件と解析フローチャートを示しますが，"メディアチェック&DPB 更新"の部分は**図 3・62** を参照してください．

ファンクション番号	1FH	32H
名称	DPBアドレスの取得	
入力条件	なし	DL…ドライブ番号 (0…カレント)
出力条件	AL…FFH 異常終了 00H 正常終了 ES:BX…DPBアドレス	

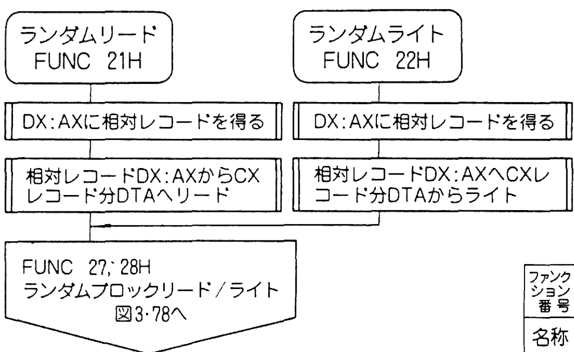
図 3・72 ファンクション **1FH,32HDPB** アドレス取得の入出力条件と解析フローチャート

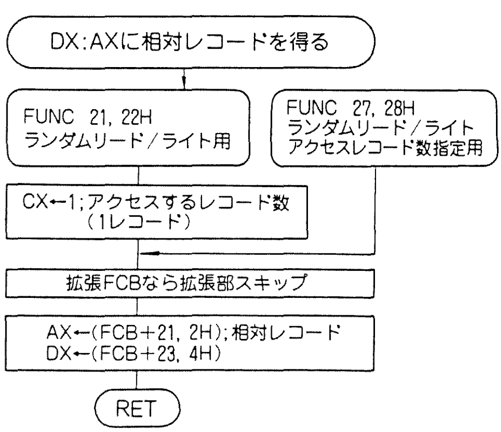
ファンクション21H
ランダムリード

ファンクション22H
ランダムライト

ランダムなファイルアクセスは相対レコードフィールド(FCB+21H～24H)を使って行われます.相対レコードフィールドがランダムリード/ライトのファイルポインタ(レコード単位)となります. 相対レコードフィールドは4バイトありますが, 最上位1バイトが0のときは最上位バイト(FCB+24H)は使用されません. 入出力条件はファンクション14, 15Hのシーケンシャルリード/ライトと同じです.

”DX : AXに相対レコードを得る”サブルーチンは, ファンクション27, 28HのCXにアクセスレコード数を指定したランダムリード/ライトと共通していて, ファンクション21, 22Hのほうで使用するエントリはCX←1としています. すなわち, 一度に1レコード分しかアクセスしないということです. 一方, ファンクション27, 28Hは一度にアクセスするレコード数をCXレジスタに指定することができます. **図3・73**に入出力条件と解析フローチャートを示します.





ファンクション番号	21H	22H
名称	ランダムリード	ランダムライト
入力条件	DS:DX…FCBアドレス	
出力条件	AL…0…正常終了 1…アクセスしたレコードにデータが入っていない 2…DTAの大きさが1レコードの大きさより小さい 3…ファイルの終わりまで読み出した(バッファの残りは0でうめた).	AL…0…正常終了 1…Disk Full 2…DTAの大きさが1レコードの大きさより小さい

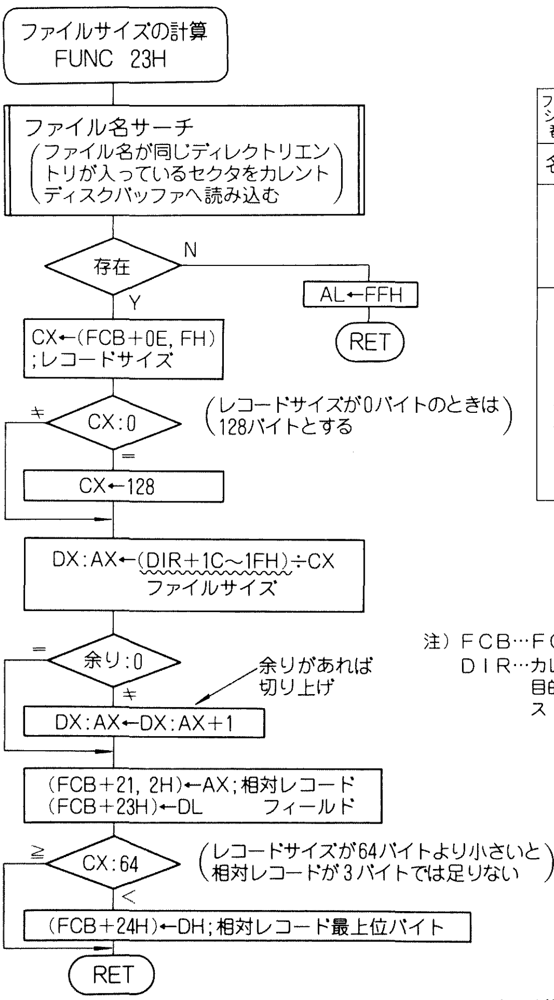
図3・73 ファンクション**21,22H**ランダムリード/ライトの入出力条件と解析フローチャート

ファンクション23H
ファイルサイズの計算

FCBで指定されたファイル名と一致するディレクトリエントリの+1C～1FHのファイルサイズ(バイト単位)から，レコード数単位のファイルサイズになおしてFCBの相対レコードフィールド(FCB+21H～24H)にセットします．すなわち，このファンクションを実行したあとでランダムなファイルライト（ファンクション22H，28H）を行えば，”アペンド”を実現できます．

FCBのレコードサイズ(FCB+0E，FH)が0のときは128バイト/レコードとみなされます．

図3・74に入出力条件と解析フローチャートを示します．



ファンクション番号	23H
名称	ファイルサイズの計算
入力条件	DS:DX…FCBアドレス
出力条件	AL…00H　正常終了 FFH　指定されたファイル名が存在しない (FCB+21～24H) 相対レコードフィールドにファイルサイズのレコード数をセット

注）FCB…FCBの先頭アドレス
DIR…カレントディスクバッファへ読み込んだ目的のディレクトリエントリ先頭アドレス

図3・74　ファンクション23H　ファイルサイズの計算の入出力条件と解析フローチャート

ファンクション24H
相対レコードフィールドのセット

シーケンシャルアクセス(ファンクション 14, 15H 等)で使う"カレントブロック(FCB+0C, DH)", "カレントレコード(FCB+20H)"からレコード単位のファイルポインタを計算し, ランダムアクセス(ファンクション 21, 22, 27, 28H)で使う"相対レコードフィールド(FCB+21H~24H)"にセットします. すなわち, シーケンシャルアクセスをしてから途中でこのファンクションを実行すれば, ランダムアクセスができるようになります.

図 3・75 に入出力条件と解析フローチャートを示します.

ファンクション番号	24H
名称	相対レコードフィールドのセット
入力条件	DS:DX…FCBアドレス
出力条件	(FCB+21~24H) 相対レコードフィールドにカレントブロックとカレントレコードから計算した値をセット

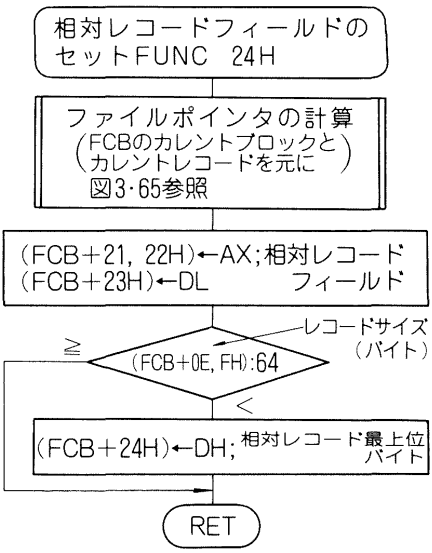


図 3・75 ファンクション **24H** 相対レコードフィールドのセットの入出力条件と解析フローチャート

ファンクション25H
割り込みベクタのセット

ファンクション35H
割り込みベクタのリード

割り込みベクタ番号とは，INT コール番号のことです．割り込みベクタとはセグメント 0 のオフセット 00～3 FFH に格納されている INT コール先アドレスのことです．INT0 がアドレス 0～3，INT 1 がアドレス 4～7…というように，1 つの INT コールで 4 バイト占有しています．この 4 バイトが INT のコール先アドレスで，オフセット（2 バイト），セグメント（2 バイト）の順で並んでいます．割り込みベクタ番号と格納アドレスの関係は，（格納オフセットアドレス）＝（割り込みベクタ番号）＊4 です．

図 3・76 に入出力条件と解析フローチャートを示します．

ファンクション番号	25H	35H
名称	割込ベクタ セット	割込ベクタ リード
入力条件	AL…割込番号 DS:DX…セットしたいアドレス	AL…割込番号
出力条件	なし	ES:BX …割込ベクタアドレス

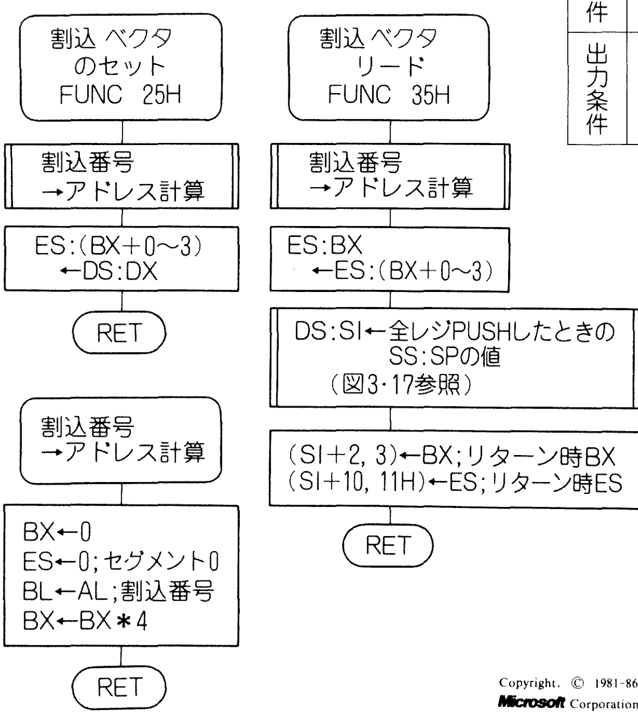


図 3・76　ファンクション 25H,35H　割り込みベクタのセット/リードの入出力条件と解析フローチャート

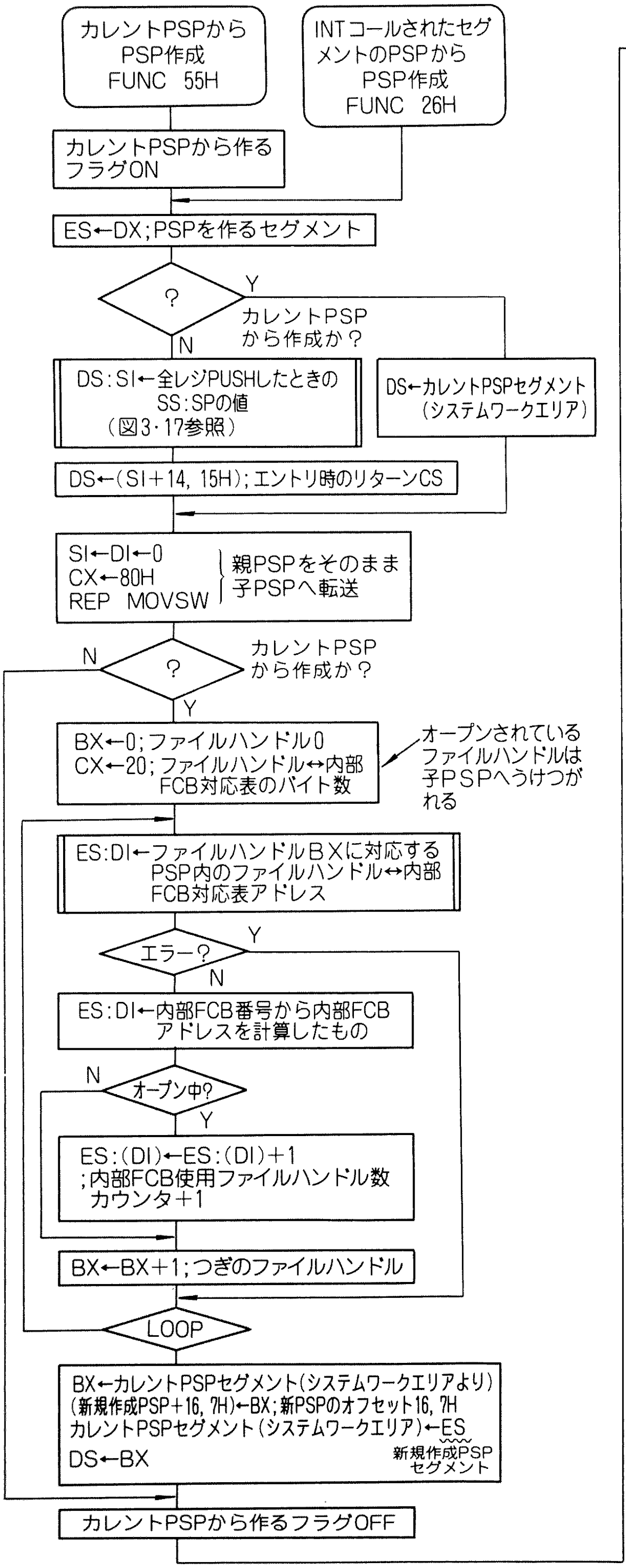
ファンクション26H
PSPの作成（INTコールされたセグメントから）

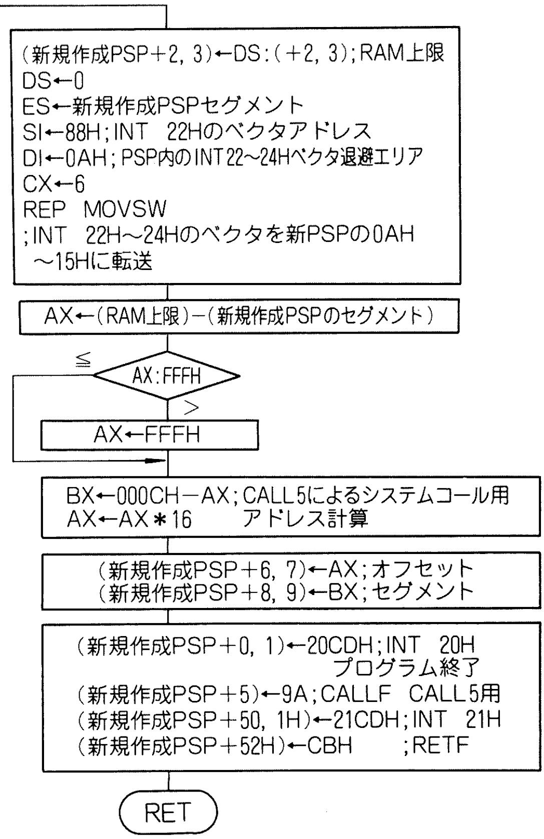
ファンクション55H
PSPの作成（カレントPSPから）

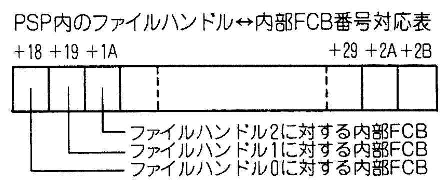
子プロセスを実行するさい，PSP を作らなければなりません．そのようなときに必要となるのがこのファンクションです．このファンクションは，親プロセスの PSP をもとにして子プロセス用の PSP を作ります．そのためには親プロセスの PSP のセグメントアドレスを指定しなければなりませんが，ファンクション 26H のほうは CS(コードセグメント)のオフセット 00～FFH に親 PSP があるものとして作成します．したがって COM 型のときに使用するものです．一方ファンクション 55H のほうは親 PSP をカレント PSP としますので，EXE 型・COM 型両方に使えます．

ファンクション 55H で PSP を作成すると，カレント PSP セグメントが新規作成した PSP のセグメントに変更されるので，それ以降カレント PSP を使ったファンクションは新規作成 PSP に対してなされるので，注意してください．あくまでも子プロセスを実行するための PSP 作成と考えてください．

図 3・77 に入出力条件と解析フローチャートを示します．







ファンクション番号	26H	55H
名称	PSP作成(COM用)	PSP作成(カレントPSPから)EXE対応
入力条件	DX…新しくPSPを作るセグメント	
出力条件	(注)ファンクション55HでPSPを作った場合,カレントPSPが新規PSPセグメントに変更される	

図3・77 ファンクション26H,55H PSP作成の入出力条件と解析フローチャート

ファンクション27H
ランダムブロックリード

ファンクション28H
ランダムブロックライト

ファンクション 21, 22H のランダムリード/ライトは，一度にアクセスできるレコード数は 1 レコードときまっていましたが，ファンクション 27, 28H では複数のレコード(CX レコード)をアクセスすることができます．

その他はファンクション 21, 22H と同じです．また，ファンクション 14, 15H(シーケンシャルリード/ライト)とこのファンクション 27, 28H は，大部分共通のルーチンを使っています．内部的には，レコード単位のファイルポインタを使うという点では共通しています．

図 3・78 に入出力条件と解析フローチャートを示します．

ファンクション番号	27H	28H
名称	ランダムブロックリード	ランダムブロックライト
入力条件	DS:DX←FCBアドレス CX←アクセスするレコード数	
出力条件	CX←実際アクセスしたレコード数 AL←0…正常終了 1…レコード内にデータがない(リード) Disk Full (ライト) 2…DTAの領域不足 3…ファイルの終わりまで読み出した (バッファの残りは0で埋めた)	

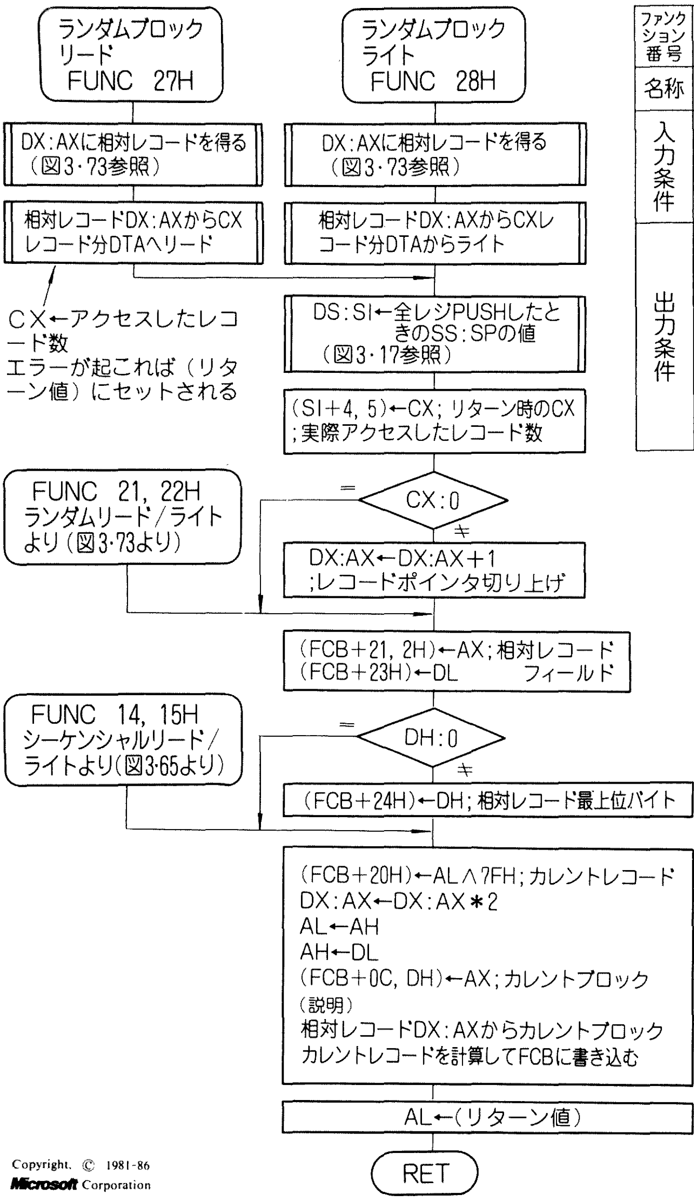


図3・78 ファンクション **27, 28H** ランダムブロックリード/ライトの入出力条件と解析フローチャート

ファンクション29H
ファイル名解析

ALレジスタの指示(各ビットに意味を持たせてある)にしたがって，DS：SIが指す文字列を解析して，ES：DIが指すFCB用バッファにFCB用のファイル名を作成するファンクションです．

ALの制御ビットの意味は

bit0……ファイル名の前のファイル区切り記号(20H以下のコントロールコード，
．”？［］：<｜>＋＝;，タブ，空白のこと)を
1　無視して解析を続ける
0　検出したら解析を中断する

bit1……デフォールトのドライブ番号指定
ファイル名にドライブ番号がないとき
0　カレントドライブ0が入れられる
1　FCBのドライブ部はそのまま

bit2……デフォールトのファイル名
ファイル名(8バイトの部分)がないとき
0　空白で埋める
1　FCBのファイル名部はそのまま

bit3……デフォールトの拡張子
拡張子(3バイトの部分)がないとき
0　空白で埋める
1　FCBの拡張子部はそのまま

ファンクション番号	29H
名称	ファイル名解析
入力条件	AL…制御ビット b0…ファイル区切記号の扱い b1…デフォールトのドライブ番号(カレント) b2…デフォールトのファイル名(空白) b3…デフォールトの拡張子(空白) DS:SI←解析したい文字列先頭アドレス ES:DI←解析結果を格納するFCBアドレス
出力条件	AL←0…ワイルドカード未使用 1…ワイルドカード使用 FFH…ドライブ名が無効 DS:SI←解析したところまでのアドレス

図3・79　ファンクション**29H**　ファイル名解析入出力条件

フローチャート中”Get1バイト”の処理は，LODSBで1バイト取得したのち”a”～”z”を小文字→大文字変換し，国別情報テーブル(**図3•93**参照)のケースマッピングコールをCALLします．以下のようにフラグが設定されます．

CF(キャリーフラグ)……20H未満のコントロールコードのとき1 ELSE 0

ZF(ゼロフラグ)　……ファイル区切り記号のとき1 ELSE 0

ここでファイル区切り記号とは

．”？［］：<｜>＋＝；，　タブ　空白

のことです．

図3・79，80に，それぞれ入出力条件と解析フローチャートを示します．

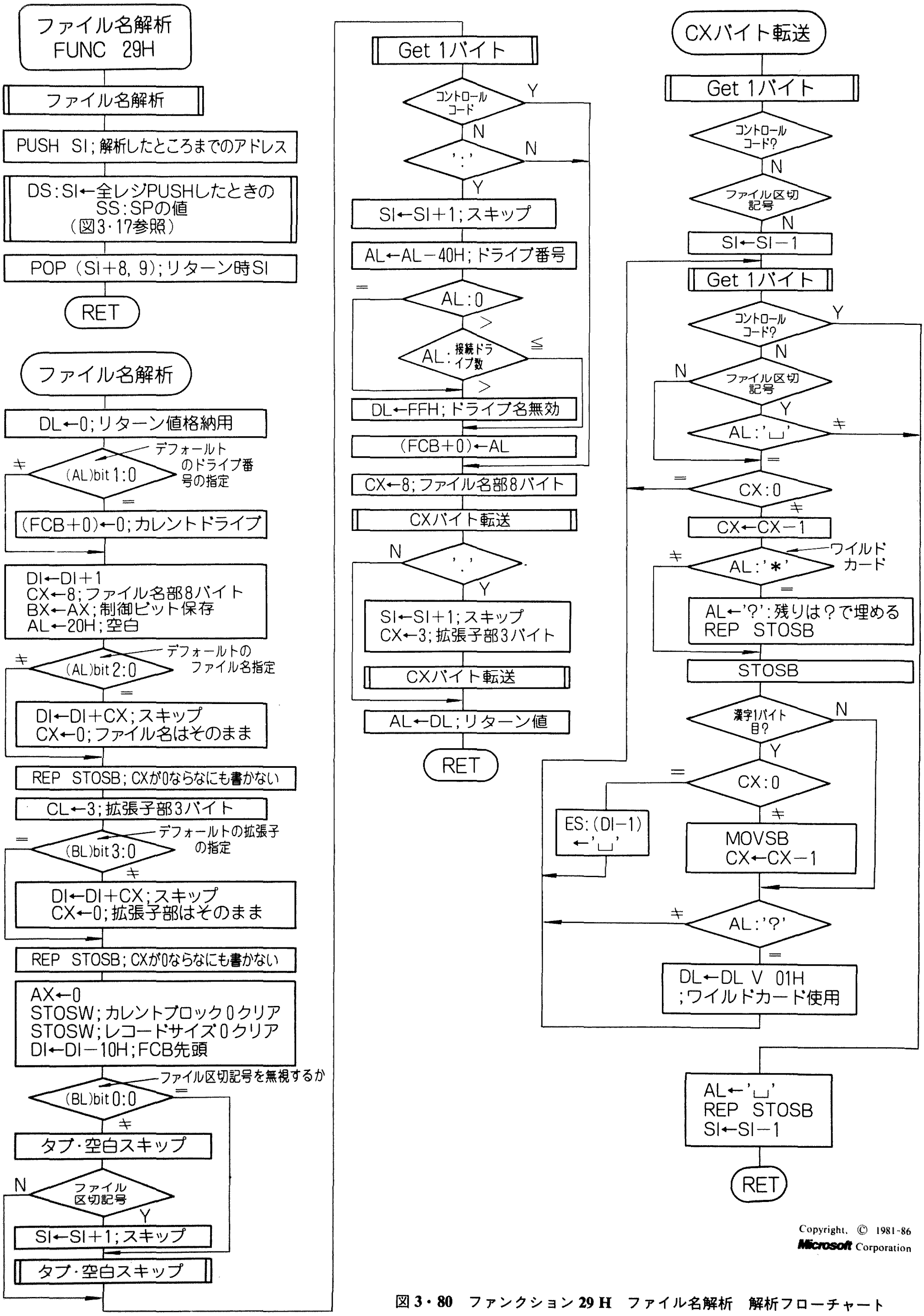


図3・80 ファンクション 29 H ファイル名解析 解析フローチャート

ファンクション2AH
日付リード

CLOCK型デバイスから日付を読み出します．各レジスタに

DH　月
DL　日
CX　年(西暦)
AL　曜日　0 1 2 3 4 5 6
　　　　　日月火水木金土

を返します．

CLOCK型デバイスはキャラクタ型デバイスの1種です．

年は西暦で1980年～2099年の120年間の範囲だけが有効です．内部的には年・月・日という形ではなく，1980年1月1日(火曜)を第1日目とした日数で日付を表現しています．CLOCKデバイスにデータ読み出しのI/Oリクエストコールコマンドを送ると6バイトの値が返ってきますが，最初の2バイトがこの1980年1月1日起算の日数となっています．あとの4バイトはそれぞれ，分・時・1/100秒・秒(時刻)です．

閏年[注)]の計算のとき単に4で割り切れるかどうかだけをチェックしているのは，1980年～2099年の間には例外の年が含まれていないためです．

図3・81，82に入出力条件と解析フローチャートを示します．

ファンクション番号	2AH	2BH
名称	日付リード	日付ライト
入力条件	なし	CX←年(西暦1980～2099) DH←月 DL←日
出力条件	CX←年 DH←月 DL←日 AL←曜日	AL←00H…正常終了 FFH…入力された日付は誤り

図3・81　ファンクション2AH，2BH
日付リード/ライトの入出力条件

注)閏年は西暦で表わしたときに4で割り切れる年であるが，例外として100で割り切れる年(ただし400で割り切れる年は除く)は閏年ではないというものである．1980年～2099年の中に100で割り切れる年は2000年であるが，2000年は400で割り切れるので閏年となり，例外の年は含まれていないのである．

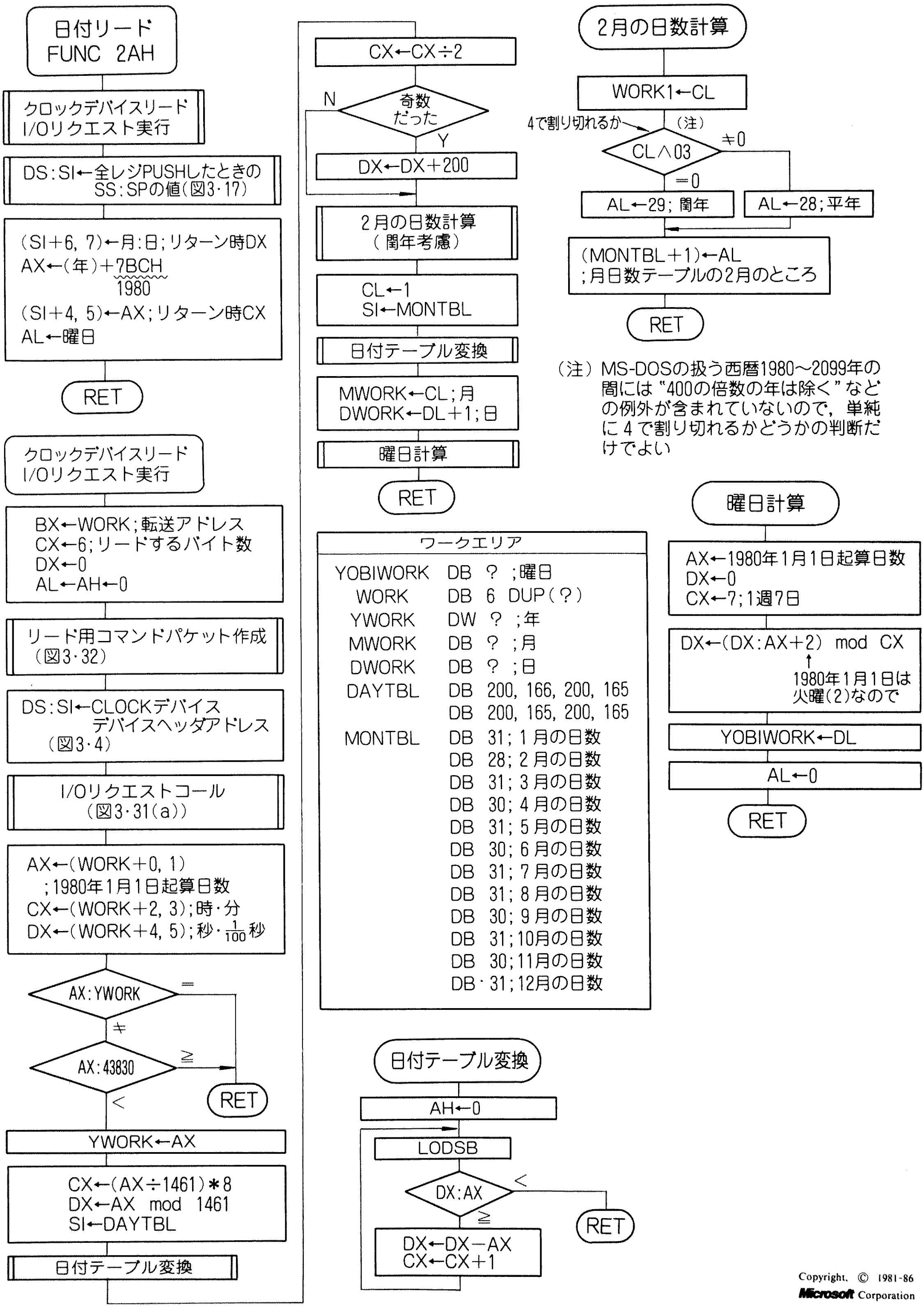


図3・82 ファンクション2AH 日付リードの解析フローチャート

ファンクション2BH
日付ライト

CLOCK デバイスのリード/ライトは日付のみのライトができないので，いったん CLOCK デバイスのリードで日付と時刻で読み出し，日付のみ書き変えてから CLOCK デバイスライトを実行します．

図 3・81，83 に入出力条件と解析フローチャートを示します．

ファンクション2CH
時刻リード

ファンクション2DH
時刻ライト

CLOCK デバイスにリード/ライトを行って実行します．CLOCK デバイスのリード/ライトは日付・時刻の組でしかできないので，時刻ライトではいったんリードして，時刻だけ書き換えてからライトしています．

CLOCK デバイスリード I/O リクエストコールを実行すると，6 バイトの値が返されますが，最初の 2 バイトは年，つぎの 4 バイトは分・時・1/100 秒・秒を表わす時刻です．

図 3・84 に入出力条件と解析フローチャートを示します．

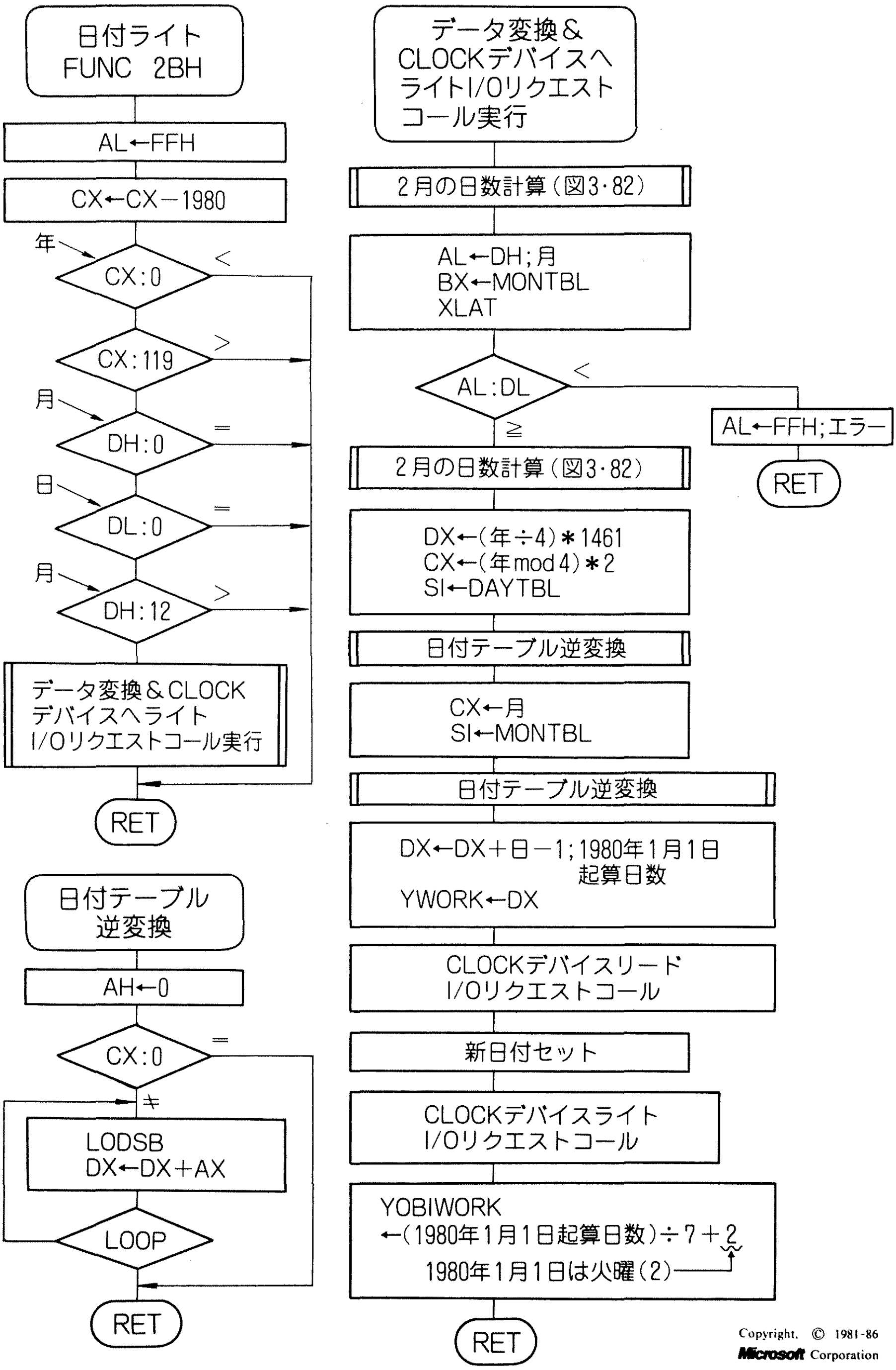


図 3・83 ファンクション 2BH 日付ライト解析フローチャート

ファンクション番号	2CH	2DH
名称	時刻リード	時刻ライト
入力条件	なし	CH←時 CL←分 DH←秒 DL←$\frac{1}{100}$秒
出力条件	CH←時 CL←分 DH←秒 DL←$\frac{1}{100}$秒 AL←0	AL←0

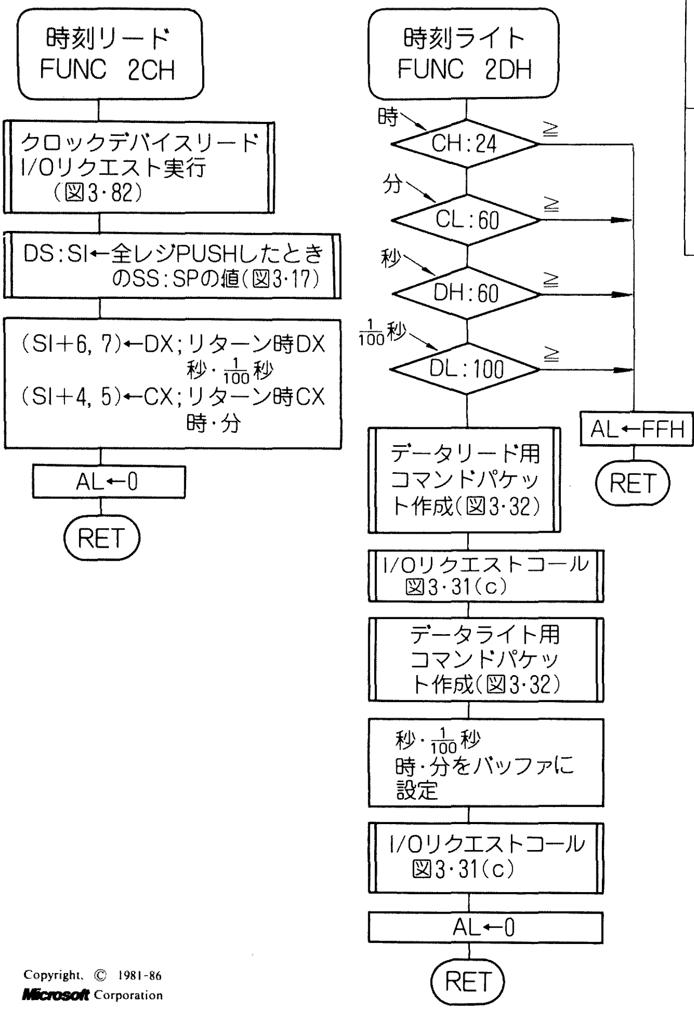


図3·84 ファンクション2CH,2DH 時刻リード/ライトの入出力条件と解析フローチャート

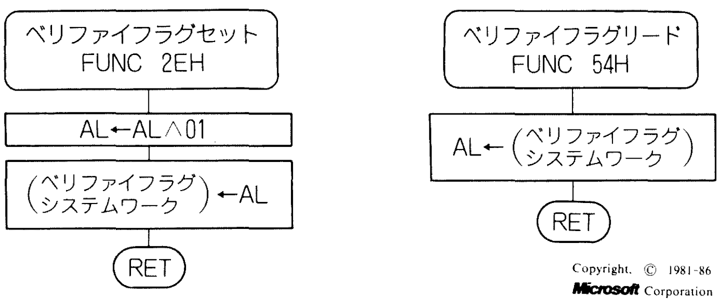
ファンクション2EH
ベリファイフラグセット

ファンクション54H
ベリファイフラグリード

ベリファイフラグとは，I/O リクエストコールでライトコマンドを発行するときに，ライト&ベリファイコマンドを実行するように指定するフラグです．

処理としては，セットのほうは単に MSDOS . SYS 内のシステムワークエリア内のベリファイフラグへ AL レジスタの値を書き込み，リードのほうは AL レジスタへ値を読み出すだけです．

図 3・85 に入出力条件と解析フローチャートを示します．



ファンクション番号	2EH	54H
名称	ベリファイフラグセット	ベリファイフラグリード
入力条件	ALのbit 0…0 ベリファイOFF 1 ベリファイON	なし
出力条件	なし	AL←ベリファイフラグ

図 3・85 ファンクション **2EH**，**54H** ベリファイフラグ セット/リード 入出力条件と解析フローチャート

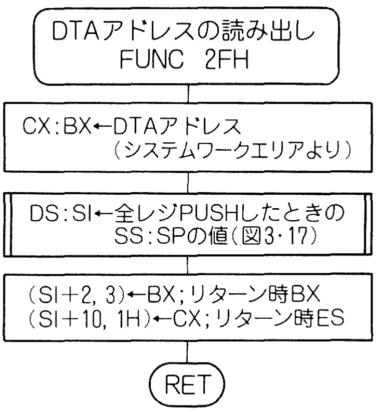
ファンクション2FH
DTAアドレスの読み出し

関連ファンクション→ファンクション 1AH"DTA アドレスの設定"

MSDOS . SYS 内のシステムワークエリアの DTA アドレスを読み出して，ES：BX にリターンするだけです．

ファンクション 1AH は DTA アドレスのセットをします．

図 3・86 に入出力条件と解析フローチャートを示します．



ファンクション番号	2FH
名称	DTAアドレスの読み出し
入力条件	なし
出力条件	ES:BX←DTAアドレス

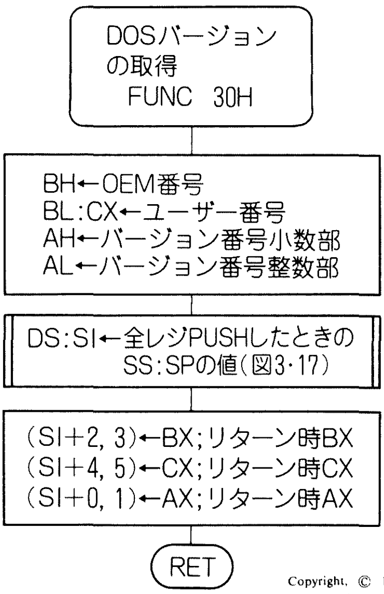
図 3・86　ファンクション 2FH　DTA アドレスの読み出し入出力条件と解析フローチャート

ファンクション30H
DOSバージョンの取得

MSDOS.SYS 内のワークエリアから MS-DOS のバージョン番号を読み出してリターンするだけです.

注意点は，バージョン番号の整数部が AL に，小数部が AH レジスタに入れられてリターンすることです.

図 3・87 に入出力条件と解析フローチャートを示します.



ファンクション番号	30H
名称	DOSバージョンの取得
入力条件	なし
出力条件	AH←バージョン番号小数部 AL←バージョン番号整数部 BH←OEM番号 BL:CX←ユーザー番号

図 3・87 ファンクション 30H DOS バージョンの取得入出力条件と解析フローチャート

ファンクション31H

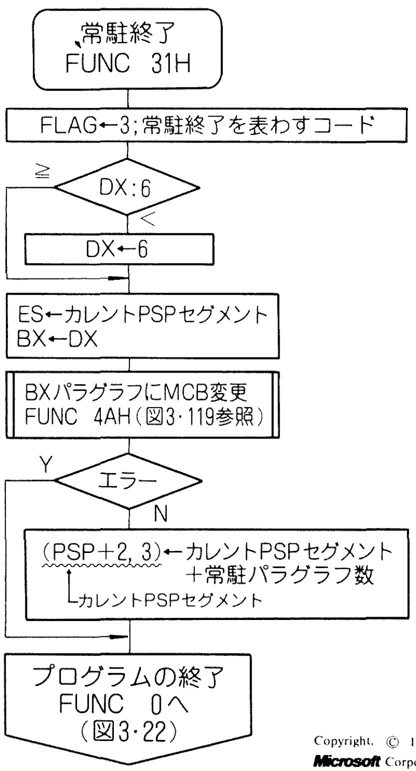
常駐終了

プロセスを終了するときに，プログラムエリアのメモリを解放せずに終了します．そのために，終了してもプロセス自体は実行できる形でメモリに残っています．

PRINT . COM などの並行処理プログラムや，MS-DOS 立ち上げ後のドライバ組み込みに使えます．

DX にメモリに常駐する部分のパラグラフ数をセットしますが，6 未満であると強制的に 6 にされます．PSP のことを考えると，少なくとも 10H パラグラフ以上にしてください．

図 3・88 に入出力条件と解析フローチャートを示します．



ファンクション番号	31H
名称	常駐終了
入力条件	AL←子プロセス終了コード DX←メモリに常駐するパラグラフ数
出力条件	なし

図 3・88　ファンクション 31H　常駐終了の入出力条件と解析フローチャート

ファンクション32H

DPBアドレス取得

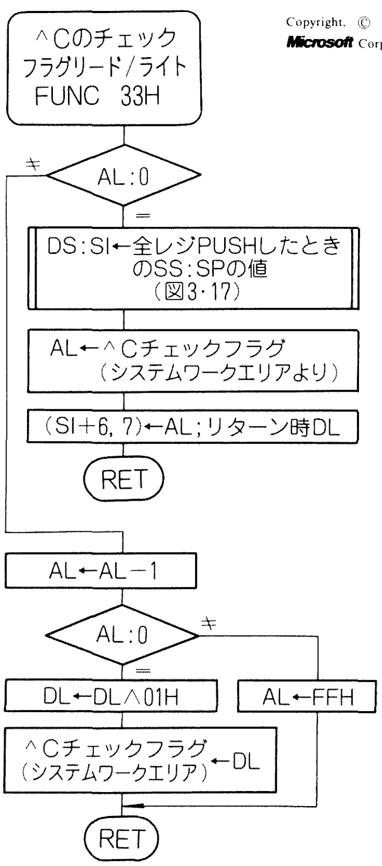
ファンクション1FH"カレントドライブのDPBアドレス取得"の項参照.

ファンクション33H

^Cチェックフラグ・リード/ライト

INT 21Hを呼び出すたびに^Cのチェックをするかどうかを指定するフラグです. ^C(STOPキーまたはCTRL+C)が押されていることをセンスすると, ^Cによる中断処理をします. ^Cによる中断処理はINT 23Hですが, MSDOS.SYS内部におけるINT 23Hの位置づけは, コラム"^Cチェック&^C処理とINT 23H"および"^C, ^S, ^P, ^Nチェック処理"(**関連図3・35, 36**)を参照してください.

図3・89に入出力条件と解析フローチャートを示します.



ファンクション番号	33H
名称	^Cのチェックフラグリード/ライト
入力条件	AL…00H フラグリード 01H フラグライト (フラグライト時) DL…bit 0…0 フラグOFF 1 フラグON
出力条件	AL…FFH…入力されたALの値が0, 1以外 (パラメータエラー) FFH以外…正常終了 (フラグリード時) DL…bit 0…0 フラグOFF 1 フラグON

図3・89 ファンクション33H ^Cのチェックフラグリード/ライトの入出力条件と解析フローチャート

ファンクション34H
INT 21H実行中フラグ（IN-DOSフラグ）アドレスの取得

”IN-DOS フラグ”とは，INT 21H エントリ時にセットされリターン時にリセットされるフラグのことで，”INT 21H を現在処理中である”ことを表わすフラグと考えたらよいのです(INT 25H，INT 26H 処理中にもセットされる)．これは MSDOS . SYS，IO . SYS がリエントラントに作られていないために，同時に 2 つ以上のプロセスから割り込みなどを使って INT 21H が実行された場合，システム内のワークエリアが壊れたり，ディスクなどのデバイスがめちゃめちゃにアクセスされたりしないようにするためです．

具体的に使われ方をあげると，PRINT . COM は割り込みで並行処理するように作られていますが，ファンクション 34H で IN-DOS フラグアドレスを取得し，INT 21H を実行する必要があると，この IN-DOS フラグを見て現在 INT 21H が他のプロセスで使われていないかを確認して，使われていないときに INT 21H を実行するのです．

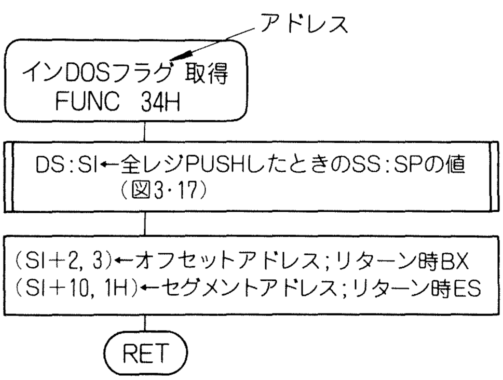
IN-DOS フラグは

0…… INT 21H は現在使用されていない

1…… INT 21H は現在他のプロセスで使用中

という意味を持っています．

図 3・90 に入出力条件と解析フローチャートを示します．



ファンクション番号	34H
名称	インDOSフラグアドレス取得
入力条件	なし
出力条件	ES:BX←インDOSフラグアドレス

図 3・90 ファンクション **34H** **INT 21H** 実行中フラグ (**IN-DOS** フラグ)
アドレスの取得の入出力条件と解析フローチャート

ファンクション35H

割り込みベクタのリード

ファンクション 25H"割り込みベクトルライト"の項を見てください．

ファンクション36H

ディスク残り容量取得

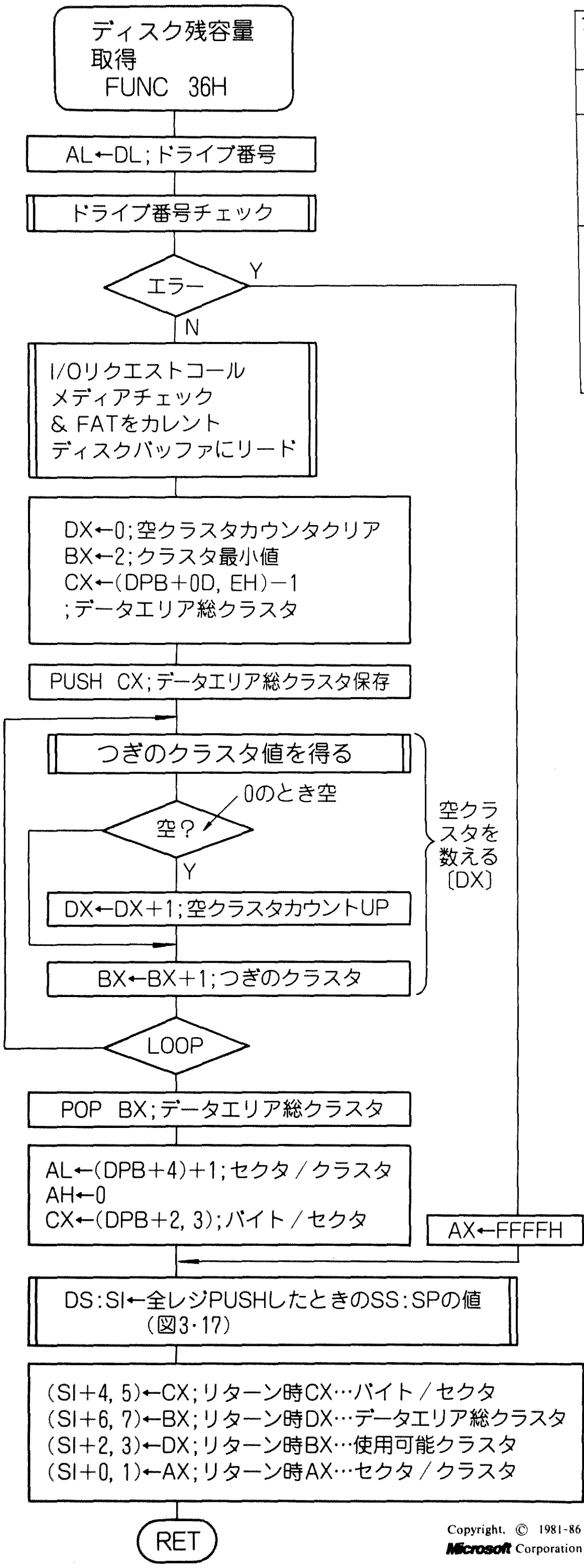
I/O リクエストコールのメディアチェックをして，FAT をカレントディスクバッファに読み込んで空クラスタ数を数えます．バイト/セクタとセクタ/クラスタ，クラスタ/ドライブの値は DPB のものをそのままリターンします．

これでディスク残り容量は

BX ...... クラスタ
AX * BX ...... セクタ
AX * BX * CX ...... バイト

で計算できます．

図 3・91 に入出力条件と解析フローチャートを示します．



ファンクション番号	36H
名称	ディスク残容量取得
入力条件	DL…ドライブ番号 (0…カレント 1…A, 2…B…)
出力条件	AX…FFFH　ドライブ番号誤り その他…セクタ/クラスタ BX…使用可能クラスタ数 CX…バイト/セクタ DX…データエリア総クラスタ数

図3・91　ファンクション 36H　ディスク残り容量取得の入出力条件と解析フローチャート

ファンクション37H
パラメータセパレータ・デバイス名許可フラグのリード/ライト

DOSのバージョン3.0以上では”デバイス名許可フラグのリード/ライト”のほうはなくなっています.

図3・92に入出力条件と解析フローチャートを示します.

ファンクション番号	37H
名称	パラメータセパレータ デバイス名許可フラグのリード/ライト
入力条件	AL…0　パラメータセパレータの取得 1　パラメータセパレータの設定 2　デバイス名許可フラグの取得 3　デバイス名許可フラグの設定 (設定のとき) DL…{ ・パラメータセパレータ ・デバイス名許可フラグ
出力条件	AL…FFH　パラメータエラー (取得のとき) DL…{ ・パラメータセパレータ ・デバイス名許可フラグ

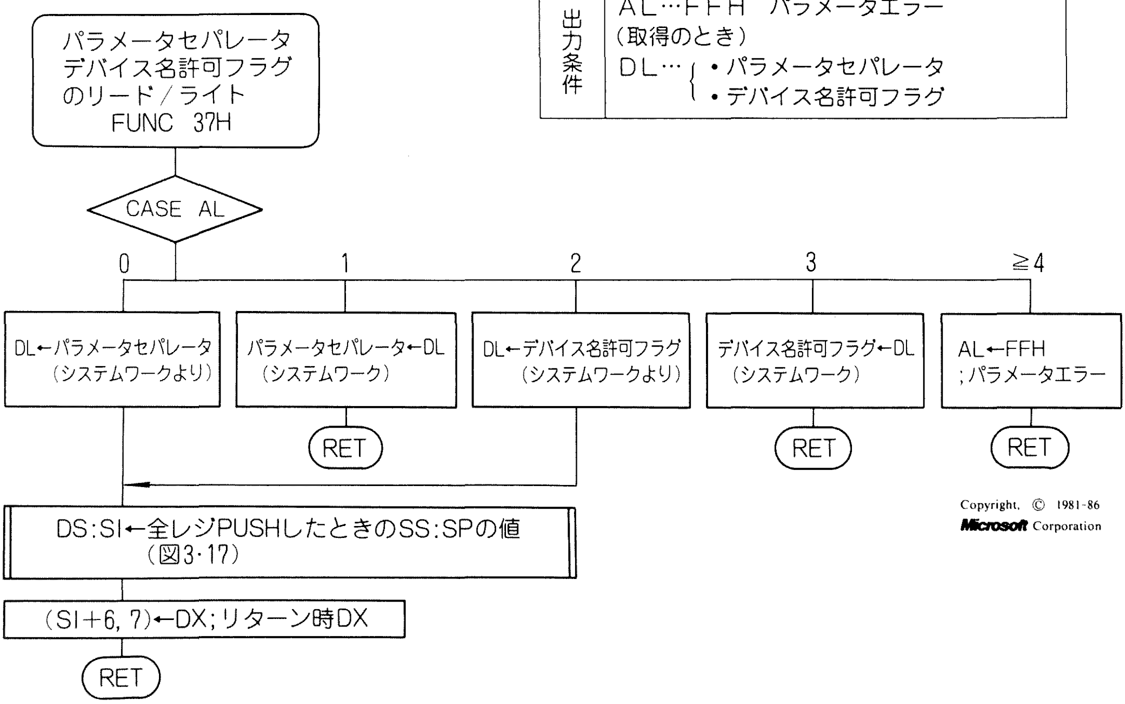


図3・92　ファンクション37H　パラメータセパレータ・デバイス許可フラグリード/ライトの入出力条件と解析フローチャート

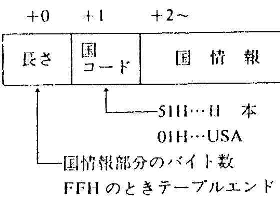
ファンクション38H

国別情報の取得

時刻や日付の表示法，数字の区切り記号など，国によって異なるこれらの情報を記録しているテーブルを国別情報テーブルといいます．国別情報テーブルには，**図3・93**のような情報が記録されています．

国情報としては，MSDOS.SYS内にあらかじめ記録されているUSA(01H)，日本(51H)の中からしか選べません．しかし，国情報テーブルアドレスを取得し**図3・93**の値に変更すれば，たとえば日付・時刻表示フォーマットを1に変更してやればヨーロッパ形式で表示するようにできますし，日付・時刻などの分離記号も変えることができます．

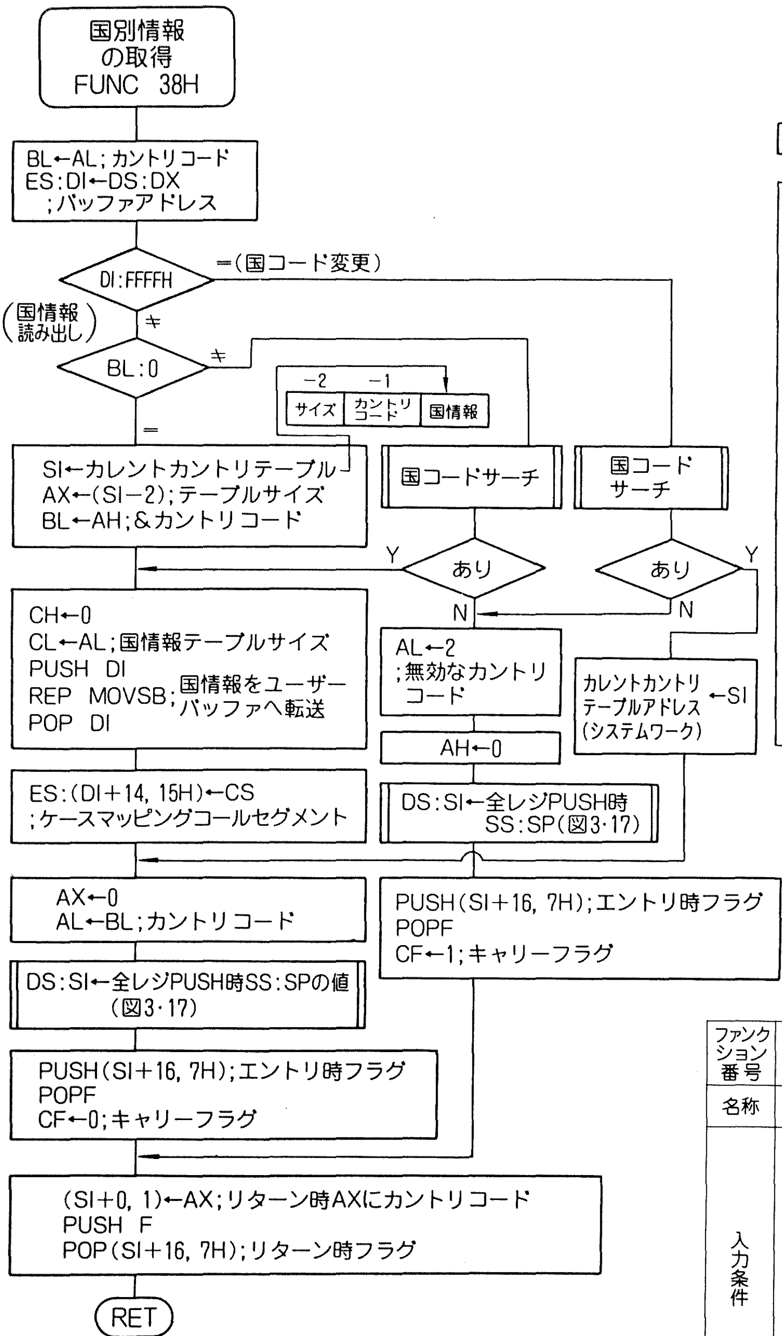
図3・94，95に入出力条件と解析フローチャートを示します．



国情報

オフセット	内容	USA	日本	
+0，1	日付時刻表示フォーマット	0 時:分:秒 月/日/年 で表示 (1のときヨーロッパ形式 時:分:秒 日/月/年 で表示)	2 年/月/日 時:分:秒 表示	
+2～+6	通貨記号	'$'0000	'¥'0000	ASCIZ文字列
+7，8	3桁ごとの区切り記号	','0	','0	ASCIZ文字列
+9，AH	10進分離記号	'.'0	'.'0	ASCIZ文字列
+0B，CH	日付分離記号	'−'0	'−'0	ASCIZ文字列
+0D，EH	時刻分離記号	':'0	':'0	ASCIZ文字列
+0FH	通貨記号の位置 bit0 …0 金額の前 1 金額の後 bit1 …0 くっつける 1 空白を入れる	00H	00H	
+10H	通貨金額の小数点以下の桁数	2	0	
+11H	時刻フォーマット 0…12時制，1…24時制	0	1	
+12～15H	ケースマッピングコールアドレス	RET命令を指している すなわち，無処理	同左	
+16，17H	データリスト分離記号	','0	','0	ASCIZ文字列
+18～1FH	未定義	—	—	

図3・93　国情報テーブルの内容



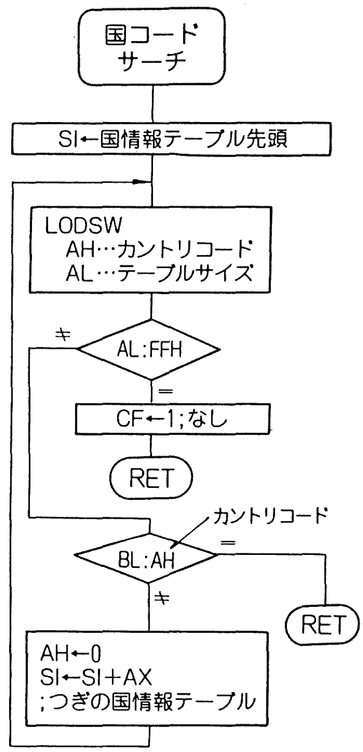


図 3・95 ファンクション 38H 国別情報の取得の解析フローチャート

ファンクション番号	38H
名称	国別情報の取得
入力条件	(読み出しの場合) AL…0 現在設定されている国情報 01H USA 51H 日本 DS:DX…情報読み出し用バッファ (32バイト必要) ただしDX≠FFFFH (設定の場合) AL…設定するカントリコード DX…FFFFH
出力条件	CF←1 異常終了 AL←2 無効なカントリコード CF←0 正常終了

図 3・94 ファンクション 38H 国別情報の取得の入出力条件

ファンクション39H
サブディレクトリの作成

図 3・96 に処理を図解したものを示しますので，見ながら読んでください．

通常のようにファイルをクリエイト(作成)するように指定されたディレクトリに新規にディレクトリエントリを作成し，空クラスタをさがし，0 クリアします．空クラスタの最初のディレクトリ名を'.'とし，最初のクラスタ値(DIR＋1A，BH)に空クラスタ番号を入れ，つぎのディレクトリ名を'..'とし，最初のクラスタ値に新規にディレクトリエントリを作成したディレクトリがあるクラスタ値(ルートディレクトリは 0)を入れます．新規に作成したディレクトリエントリの最初のクラスタ(DIR＋1A，BH)には，いま'.'と'..'を作成した空クラスタ値を入れます．

すなわち，"サブディレクトリ作成"で作られるディレクトリエントリは 3 つあって，1 つはパス名で指定されたディレクトリに，あとの 2 つは空クラスタに作られたディレクトリに子ディレクトリを表す'.'と親ディレクトリを表わす'..'です．

図 3・97 に入出力条件と解析フローチャートを示します．

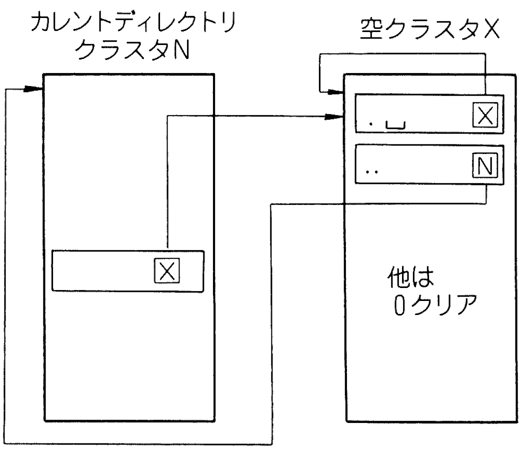
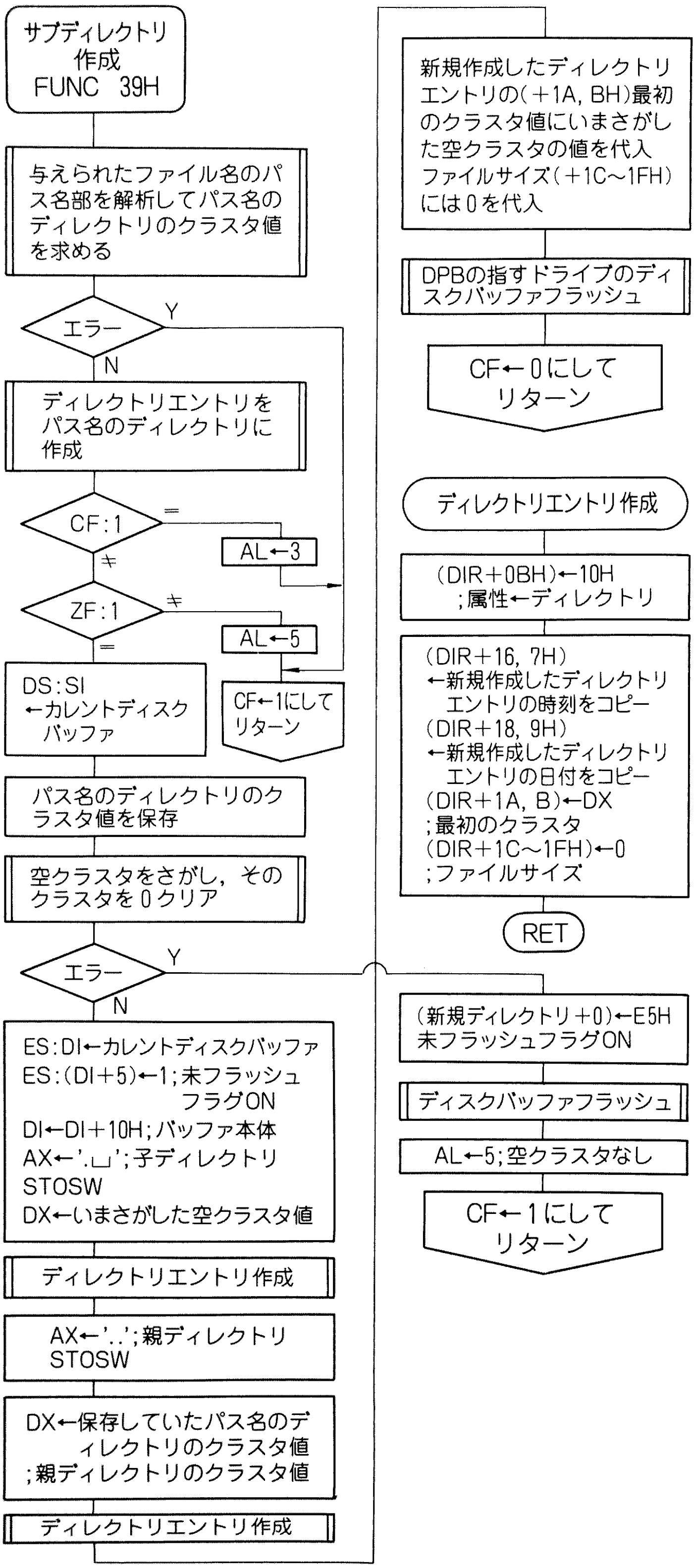


図 3・96 サブディレクトリの作成処理状況



ファンクション番号	39H
名称	サブディレクトリの作成
入力条件	DS:DX←パス名先頭アドレス
出力条件	CF←0　正常終了 　　1　異常終了 AL…エラーコード 　　3　指名されたパス名がおかしい 　　5　空ディレクトリがない． 　　　すでに同名のディレクトリエントリがある

図 3・97　ファンクション 39H　サブディレクトリの削除の入出力条件と解析フローチャート

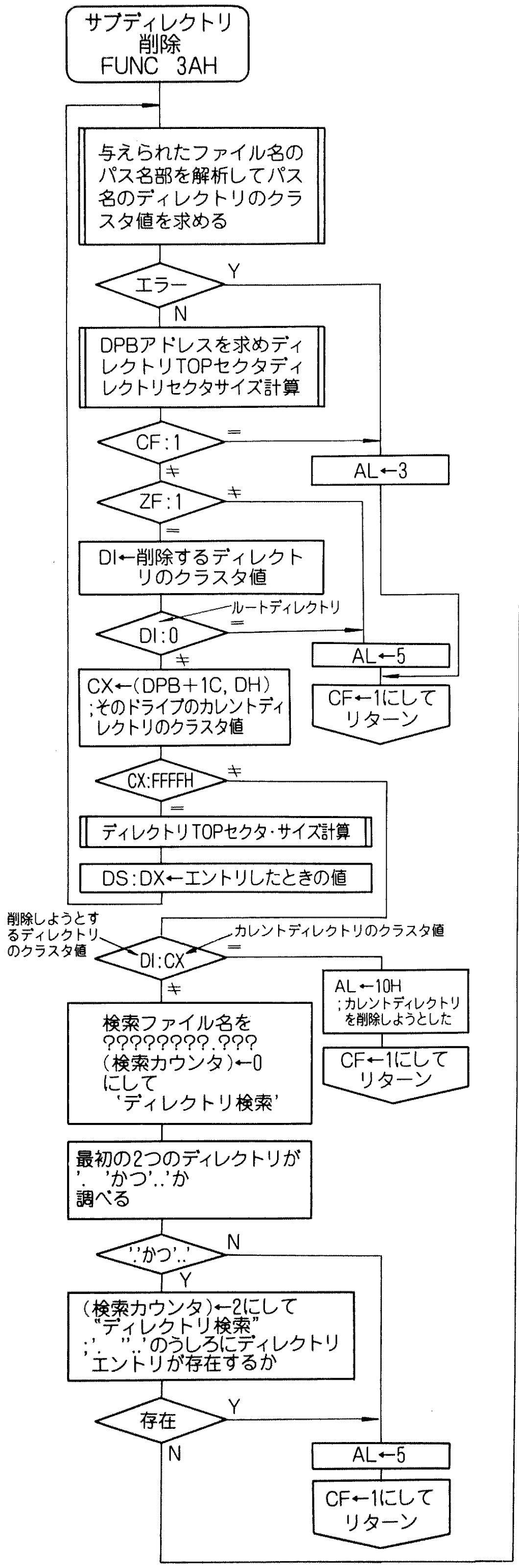
ファンクション3AH
サブディレクトリの削除

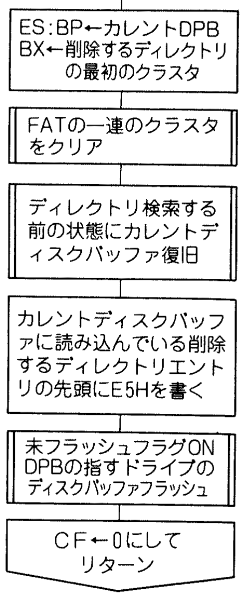
このファンクション 3AH でサブディレクトリを削除するためには

- 削除するディレクトリがサブディレクトリであること(ルートディレクトリは削除できない)
- ディレクトリエントリ中の最初の2つのディレクトリエントリ名が'.'かつ'..'であること
- ディレクトリ中に'.', '..'以外にディレクトリエントリがないこと
- 削除するディレクトリがカレントディレクトリでないこと

を満足していなければなりません．以上の事項を満足していれば，ふつうのファイル削除と同じです．

図3・98に入出力条件と解析フローチャートを示します．





ファンクション番号	3AH
名称	サブディレクトリ削除
入力条件	DS:DX←パス名アドレス
出力条件	CF←0　正常終了 　　1　異常終了 (エラーコード) AX…3 • 指定パス名がおかしい 　　　• 指定パスがない 　　5 • ルートディレクトリを削除しようとした 　　　• ディレクトリの最初の2つのファイル名が'.　'..'でない 　　　• ディレクトリ中に'.　'..'以外にディレクトリエントリがある 　10H • カレントディレクトリを削除しようとした

図3・98　ファンクション3AH　サブディレクトリの削除の入出力条件と解析フローチャート

ファンクション3BH
カレントディレクトリの変更

カレントディレクトリのパス名は，そのドライブのDPBのオフセット＋1E～5DHに記録されています．また，そのカレントディレクトリのディレクトリが記録されているクラスタ値（最初のクラスタ）がDPBのオフセット（＋1C，1DH）に格納されています．ただし，ルートディレクトリのクラスタ値は0000Hとなっています．

パス名の指定で

　　　カレントディレクトリ

　　　親ディレクトリ

を表わしますが，パス名の中で

　¥FOO¥BAR¥.¥ZONE

などと書いても'.'は無視されるだけです．また

　¥FOO¥BAR¥..¥ZONE

と書けば，'..'より1つ前の¥にポインタがもどされ，解析が続行されますので

　¥FOO¥ZONE

と書いたのと同じになります．

パス名は64バイトの制限があり，DPBのカレントディレクトリ名を格納するエリア（＋1E～5DH）は64バイト確保されています．DPBの中にカレントディレクトリ名を格納するのはバージョン2.xxで，3.xxになると別のエリアに各ドライブごとのカレントディレクトリ名をまとめて格納するようになっています．バージョン3.xxのカレントディレクトリ名格納エリアは，システムパラメータテーブル（**図3・3，図3・4**）のオフセット（＋16H～19H）にオフセット，セグメントの順にアドレスが格納されているようです．

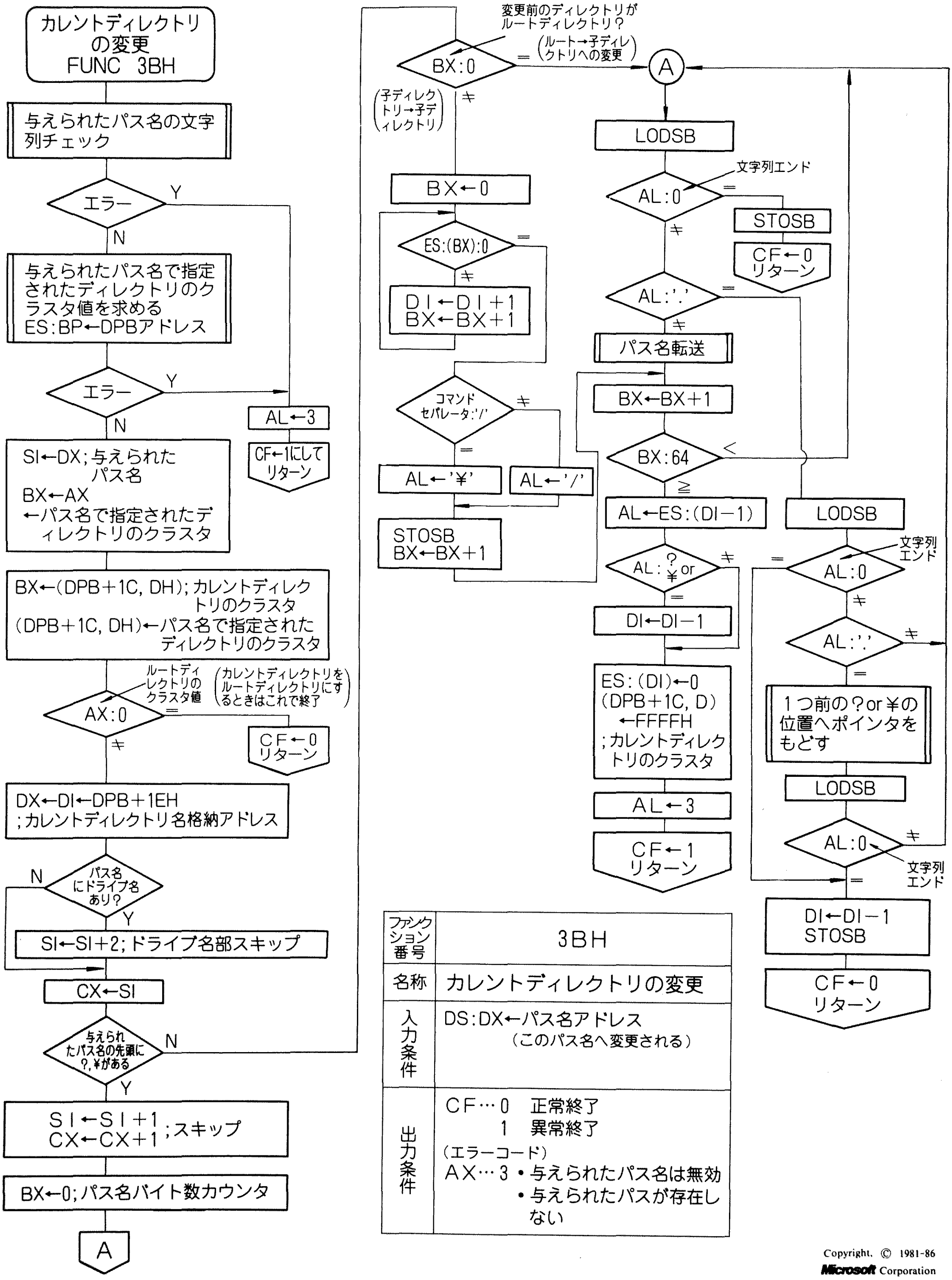
最後のパス名を入れたために

　¥FOO¥BAR¥……¥ZONE

　　　　　　　　└→64バイトを越えた

64バイトを越えたときは，カレントディレクトリのクラスタ（DPB＋1C，DH）にはFFFFHが入れられ，エラー終了します．

図3・99，100に入出力条件と解析フローチャートを示します．



ファンクション番号	3BH
名称	カレントディレクトリの変更
入力条件	DS:DX←パス名アドレス (このパス名へ変更される)
出力条件	CF…0　正常終了 　　1　異常終了 (エラーコード) AX…3 •与えられたパス名は無効 　　　•与えられたパスが存在しない

図3・99　ファンクション3BH　カレントディレクトリの変更の入出力条件と解析フローチャート

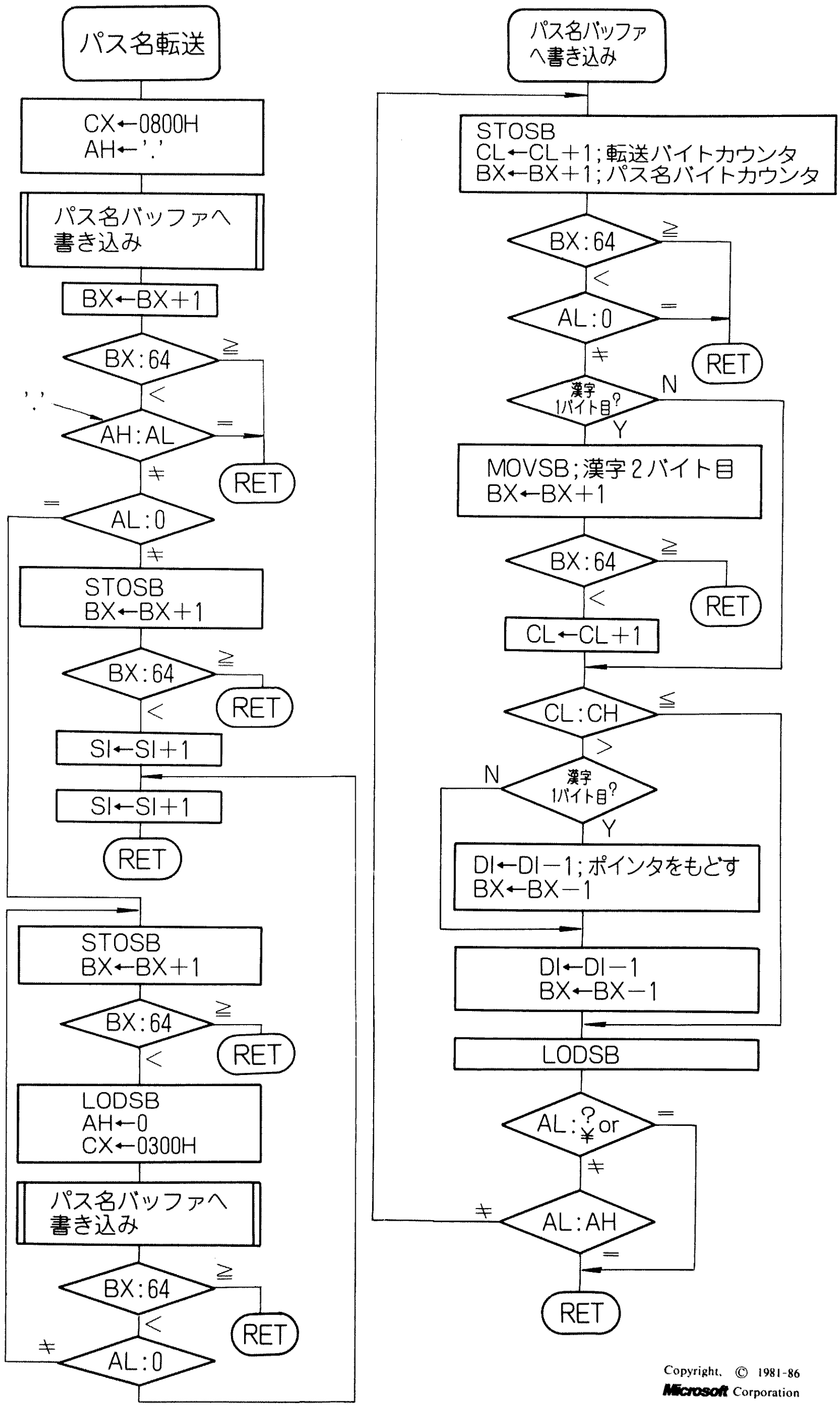


図 3・100 ファンクション 3BH カレントディレクトリの変更用サブルーチン

ファンクション3CH パス名によるファイル作成

ファンクション3DH パス名によるファイルオープン

パス名によるファイルオープンは，パス名を解析してカレントディレクトリを変更したのち，ファンクション 16H（FCB によるファイル作成）を利用してディレクトリエントリを作成します．

すなわち，同名のディレクトリエントリがすでになければ新たに作成し，あればファイルサイズを 0 にして新規ファイルとします．そのあと AL ← 2（リード/ライト　オープンモード）にしてファンクション 3DH ”パス名によるファイルオープン”へエントリします．

ファンクション 3CH では，CX レジスタに与えられたファンクション属性を新規ディレクトリエントリに書き込みますが，それが ”リードオンリ” 属性でもリード/ライトオープンとしてリターンします（もちろんキャラクタ型ディレクトリならファイル属性は無視される）．くわしくは，**図 3・102** の解析フローチャートを見てもらえばわかります．

ファンクション 3DH では，リードオンリ属性のディレクトリエントリはライトまたはリード/ライトオープンできません．

内部 FCB の先頭には 3 バイト（ただしバージョン 2. xx）のヘッダがついていて，そのあとに通

ファンクション番号	3CH	3DH
名称	パス名によるファイル作成	パス名によるファイルオープン
入力条件	DS：DX←パス名アドレス CX←作成したファイルにつける属性 1…リードオンリ 2…不可視 4…システムファイル 8…ボリュームラベル 10H…ディレクトリ 20H…保存ビット	DS：DX←パス名アドレス AL←オープン形式 0…リードオンリ 1…ライトオンリ 2…リード／ライト
出力条件	CF←0　正常終了 AX…ファイルハンドル CF←1　異常終了 （エラーコード） AX：2…ファイルが存在しない（FUNC　3DHのみ） 3…与えられたパス名はパス名としておかしい パスが存在しない 4…空ファイルハンドルがなかった 空内部FCBがなかった （オープンされているファイルが多すぎる） 5…ディレクトリorボリュームラベルをオープンしようとした リードオンリファイルをライト（1），リード／ライト（2）でオープンしようとした 0CH…ALの値が 0～2 以外（FUNC　3DHのみ）	

図 3・101
ファンクション **3CH**　パス名による　ファイル作成
ファンクション **3DH**　パス名による　ファイルオープン
の入出力条件

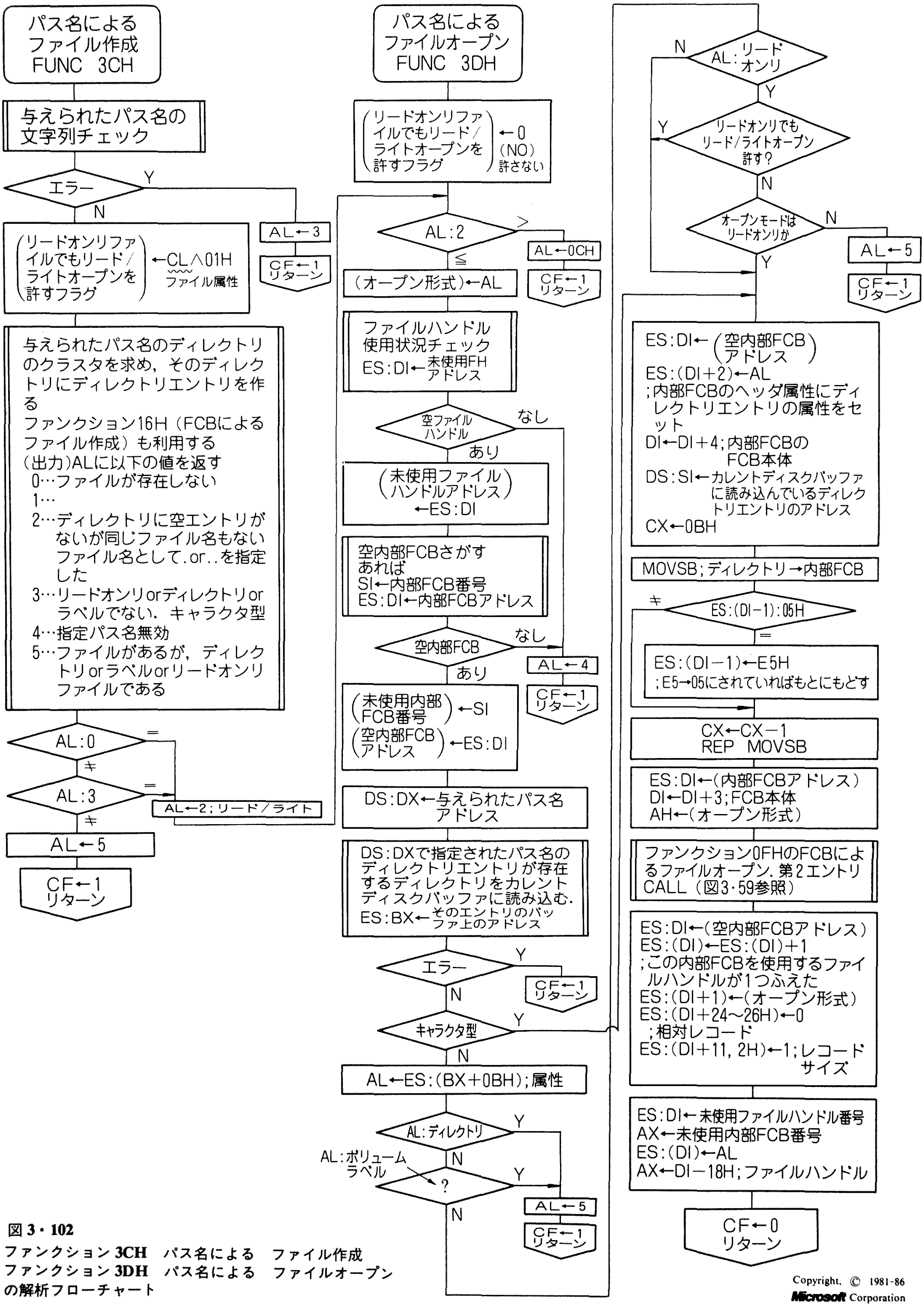


図 3・102
ファンクション 3CH パス名による ファイル作成
ファンクション 3DH パス名による ファイルオープン
の解析フローチャート

常の FCB と同じものが書かれています．ヘッダの 3 バイトは

+0 …… この内部 FCB を使っているファイルハンドル数

+1 …… オープン形式 0 リードオンリ

1 ライトオンリ

2 リード/ライト

+2 …… ファイル属性

の意味を持っていますが，オープンが成功すると（+0）の内部 FCB 使用ファイルハンドル数が+1 され，オープン形式（+1）に AL で指定されていたオープンコードを書き込みます．

ファンクション 3CH は 1 回ディレクトリをアクセスして新規ディレクトリエントリを作り，ファンクション 3DH へそのあとのオープン処理をまかせて，もう一度ディレクトリをアクセスさせますが，ディスクバッファリングがうまく働けば，ファンクション 3DH のためにディレクトリアクセスはディスクバッファメモリ上にバッファリングされたディレクトリ上で行われますので，ディスクのアクセスは 2 度手間にはなりません．

図 3・101，102 に入出力条件と解析フローチャートを示します．

ファンクション3EH
ファイルハンドルによるファイルクローズ

ファイルハンドルから内部 FCB アドレスを求め，内部 FCB(+0)のこの内部 FCB を使用しているファイルハンドル数を−1 します．内部 FCB 自体はこの値が 0 にならないかぎり消去されません(**図 3・9〜11** 参照)．1 つの内部 FCB は複数のファイルハンドルから使用されることがあるのです．

実際のクローズ処理は，内部 FCB の FCB 本体アドレスを"FCB によるファイルクローズ(ファンクション 10H) "に渡して実行します．ファンクション 10H からのリターン値が AL＝FFH のときファンクション 3EH のリターン値 AX=2 となります．

図 3・103 に入出力条件と解析フローチャートを示します．

ファンクション番号	3EH
名称	ファイルハンドルによる ファイルクローズ
入力条件	BX←ファイルハンドル
出力条件	CF←0　正常終了 　　1　異常終了 （エラーコード） AX←2　ファイルが存在しない 　　6　指定ファイルハンドルはオープンされていなかった.

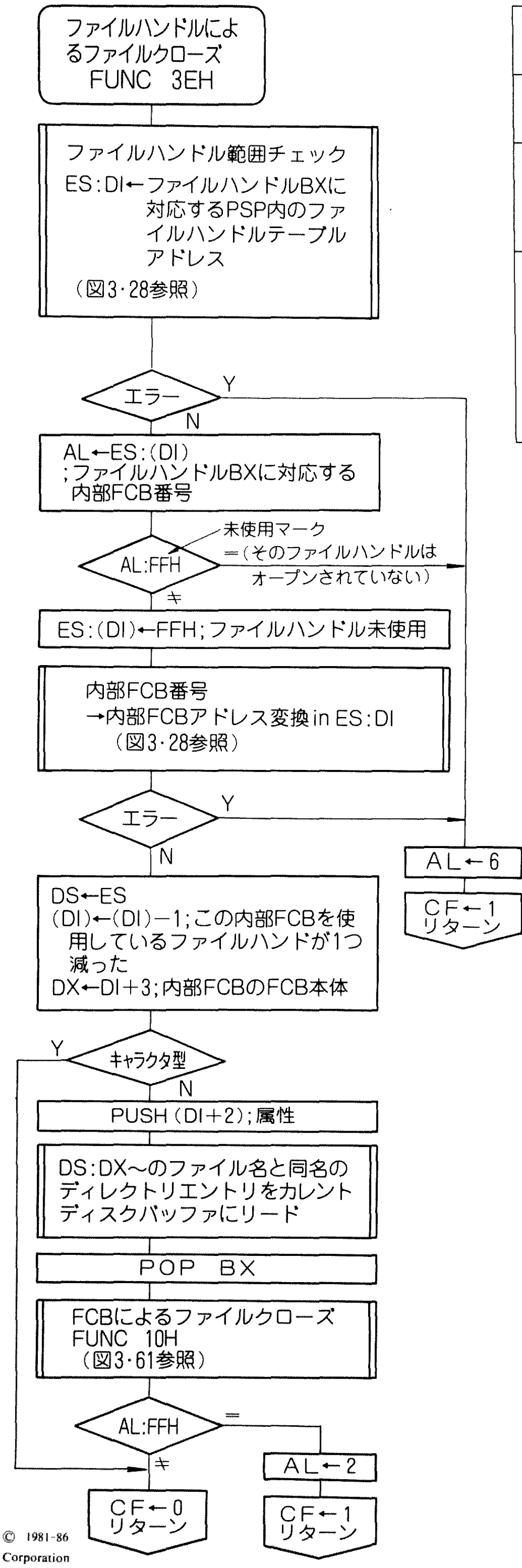


図3・103　ファンクション3EH　ファイルハンドルによるファイルクローズの入出力条件と解析フローチャート

ファンクション3FH
ファイルハンドルによるファイル/デバイスリード

ファンクション40H
ファイルハンドルによるファイル/デバイスライト

ファイルハンドルから内部FCBアドレスを求めます．リードに対してライトオンリファイル，ライトに対してリードオンリファイルをアクセスした場合には，エラー5となります．ライトオンリ/リードオンリの属性は内部FCBの（+1）で判断します（**図3・9〜11**参照）．

内部FCBのオフセット（+1）はファイルオープン属性で

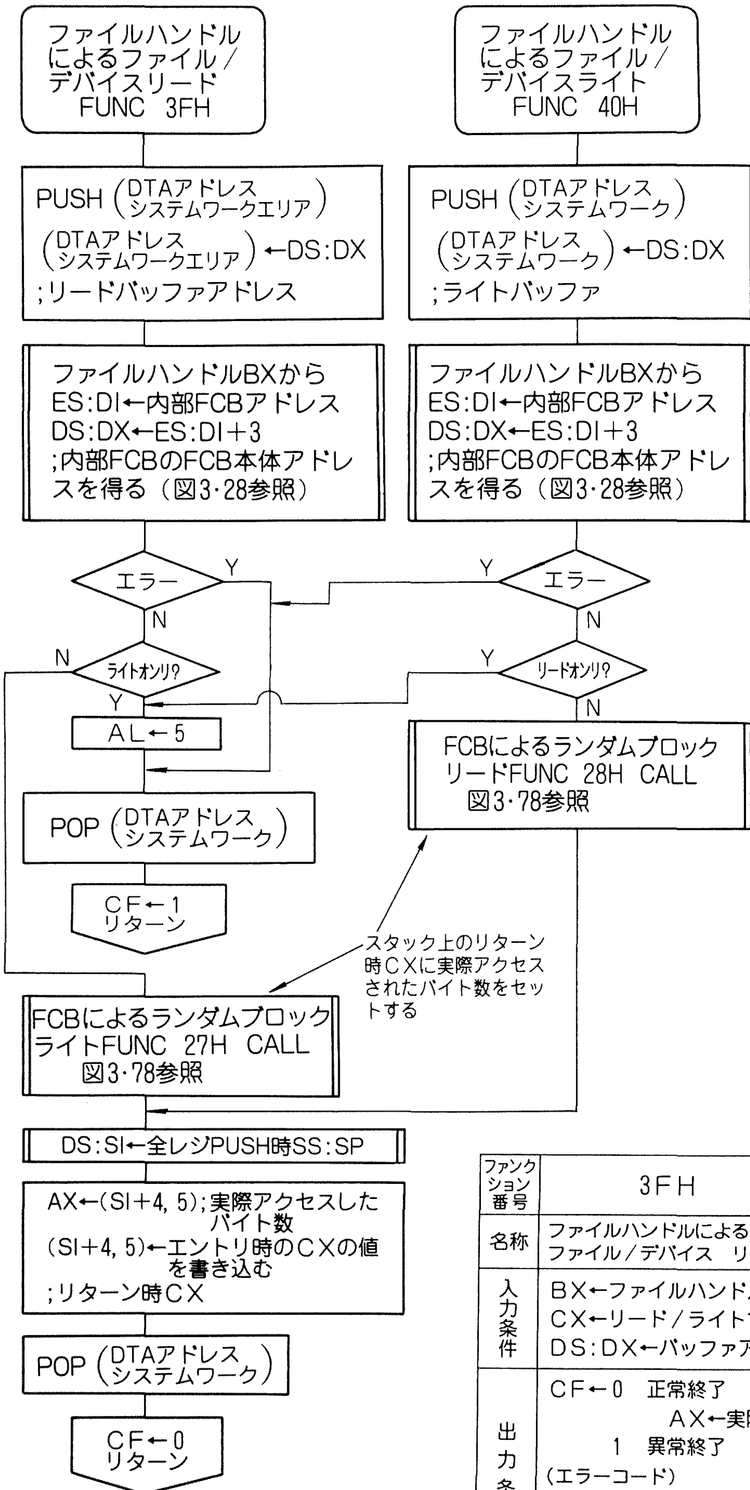
0……　リードオンリ
1……　ライトオンリ
2……　リード/ライト

となっています．この値はパス名によるオープン（ファンクション3DH）でCXレジスタに指定した番号と同じものです．

実際の処理は"FCBによるランダムブロックリード/ライト（ファンクション27H/28H）"をCALLして行います（**図3・78**参照）．

ファンクション27H/28HをCALLすると，スタック上のINT 21HエントリしたときのCXレジスタの値を実際アクセスしたバイト数に変えてしまうので，実際アクセスしたバイト数はAXレジスタに移し，スタック上のCXレジスタの値はエントリしたときのもとの値にもどします．

図3・104に入出力条件と解析フローチャートを示します．



ファンクション番号	3FH	40H
名称	ファイルハンドルによるファイル/デバイス リード	ファイルハンドルによるファイル/デバイス ライト
入力条件	BX←ファイルハンドル CX←リード/ライトするバイト数 DS:DX←バッファアドレス	
出力条件	CF←0 正常終了 AX←実際アクセスしたバイト数 1 異常終了 (エラーコード) AX←5 ライトオンリ/リードオンリだった 6 指定ファイルハンドルはオープンされていない	

図3・104 ファンクション3FH/40H ファイルハンドルによるファイル/デバイス リード/ライトの入出力条件と解析フローチャート

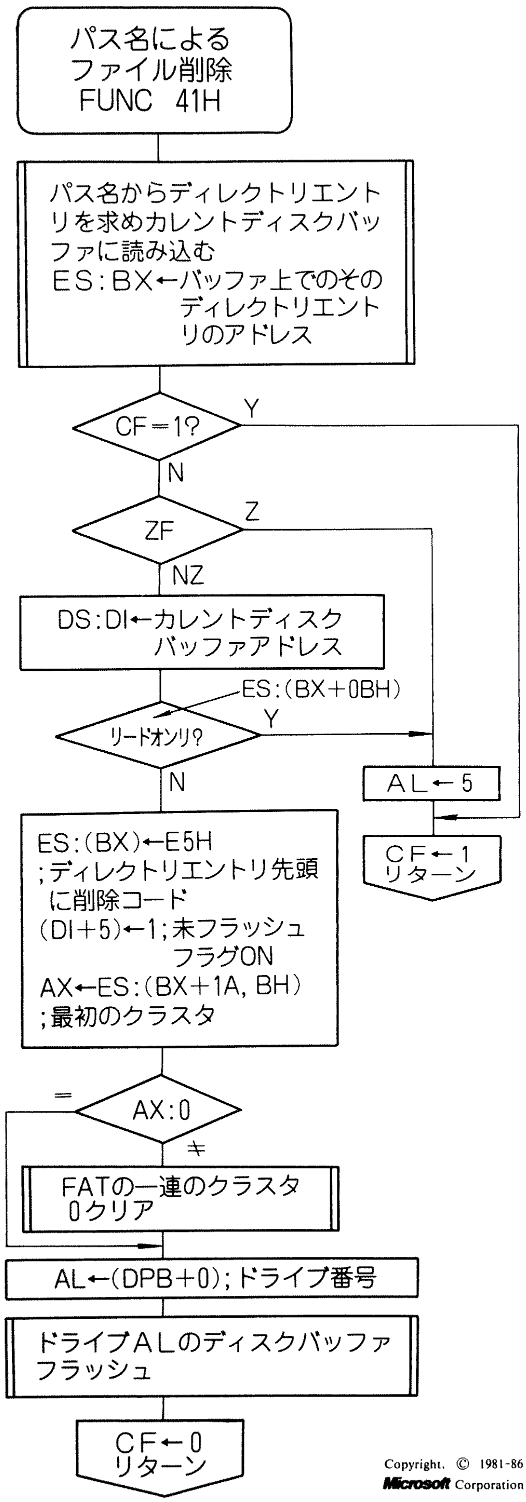
ファンクション41H
パス名によるファイル解析

パス名のパス部分から目的のディレクトリのクラスタ値を求め，目的のディレクトリからパス名のファイル名部分と同名のディレクトリエントリをさがし出し，ディレクトリエントリ先頭に削除記号 E5H を書き込むだけです．このサーチのためにはカレントディスクバッファを使用します．

パス名の例
¥FOO¥BAR¥KAISEKI¥EXE2BIN.EXE
|←――――パス部――――→|←ファイル名部→|

ディレクトリエントリオフセット（＋1A，BH）の最初のクラスタが 0000H のときは，ディレクトリエントリだけ存在し，ディレクトリ上のデータエリアは使用していなくて，FAT 上に展開されていないことを示します．

図 3・105 に入出力条件と解析フローチャートを示します．



ファンクション番号	41H
名称	パス名によるファイル削除
入力条件	DS:DX←パス名アドレス
出力条件	CF←0　正常終了 　　1　異常終了 (エラーコード) AX←2　ファイルが存在しない 　　3　与えられたパス名がパス名としておかしい 　　　　パスが存在しない 　　5　ディレクトリを削除しようとした. 　　　　リードオンリファイルを削除しようとした

図 3・105　ファンクション 41H　パス名によるファンクション削除の入出力条件と解析フローチャート

ファンクション42H

ファイルハンドルによるファイルポインタの移動

ファイルハンドルから対応する内部 FCB アドレスを求めます (**図 3•28** 参照). AL レジスタの指定にしたがって, "相対レコードフィールド (FCB オフセット＋10H～13H) "を CX：DX バイト分移動します.

AL： 0 …… ファイル先頭から CX：DX バイト分
　　 1 …… 現在位置から CX：DX バイト分
　　 2 …… ファイル終りから CX：DX バイト分

移動します. それぞれの計算法は

AL： 0 のとき CX：DX そのまま
　　 1 のとき CX：DX＋相対レコードフィールドの値 (FCB＋21H～24H)
　　 2 のとき CX：DX＋ファイルサイズ (FCB＋10H～13H)

としたものを (FCB＋21H～24H) の相対レコードフィールドに書き込みます.

キャラクタ型に対しては, CX：DX はつねに 0000：0000H で, AL←0 (ファイル先頭) として扱います.

図 3・106 に入出力条件と解析フローチャートを示します.

ファンクション番号	42H
名称	ファイルハンドルによる ファイルポインタの移動
入力条件	AL←0…ファイル先頭から 1…現在位置から 2…ファイル終わりから TOF →0 →1 →2 EOF BX←ファイルハンドル CX:DX←オフセット （ALの値にしたがって足される）
出力条件	CF←0 正常終了 DX:AX…移動後のファイルポインタのファイル先頭からのオフセット CF←1 異常終了 （エラーコード） AX←1 与えられたALの値が0～2外 6 与えられたファイルハンドルがオープンされていない

（注）TOF:Top Of File
EOF:End Of File

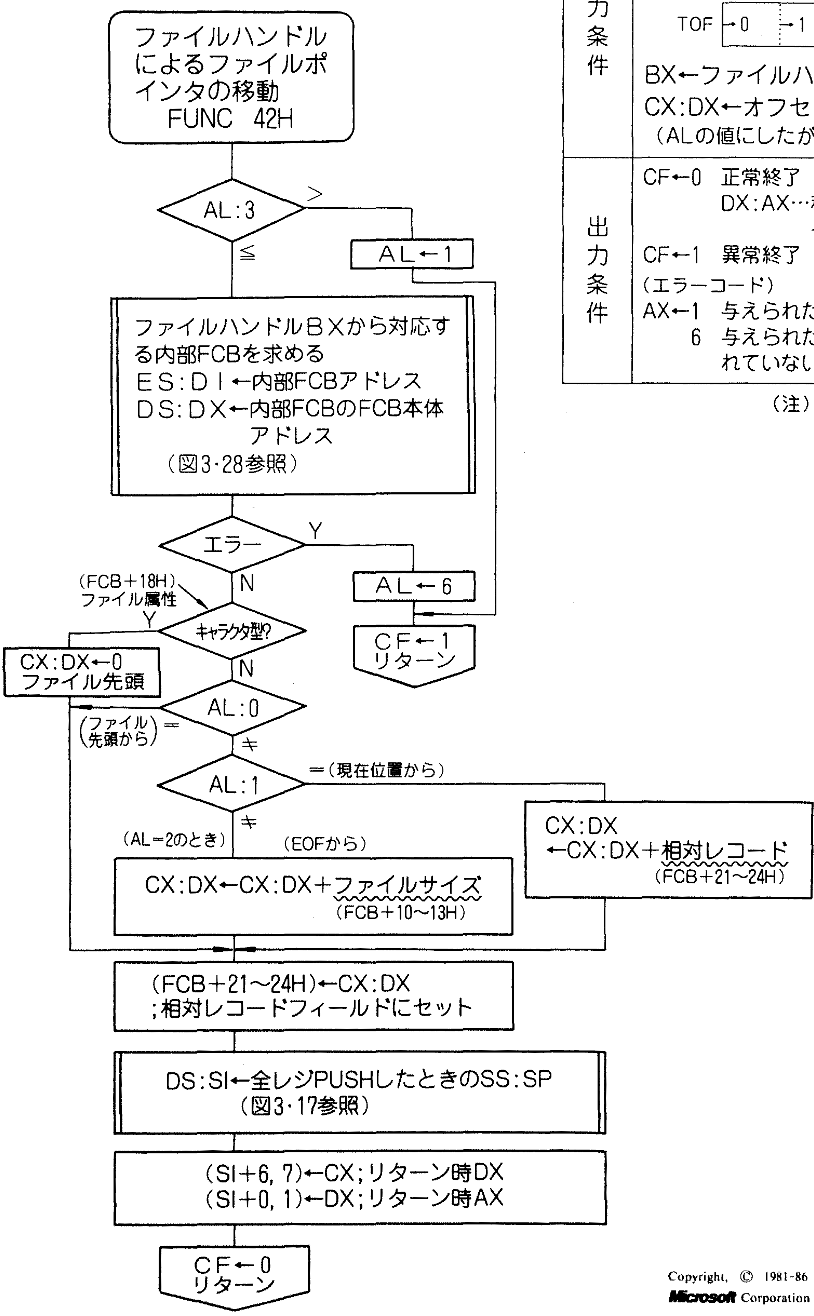


図3・106 ファンクション42H ファイルハンドルによるファイルポインタ移動の入出力条件と解析フローチャート

ファンクション43H

パス名によるファイル属性の変更

変更したい属性をCXレコードにセットしますが，変更できるのは

bit0　リードオンリ

bit1　不可視（DIRをしても表示されないetc.）

bit2　システムファイル

bit5　保存ビット

だけであって，他のビットを1にしていた場合はエラー5がリターンされます．変更できない属性としては

bit3　　　　ボリュームラベル

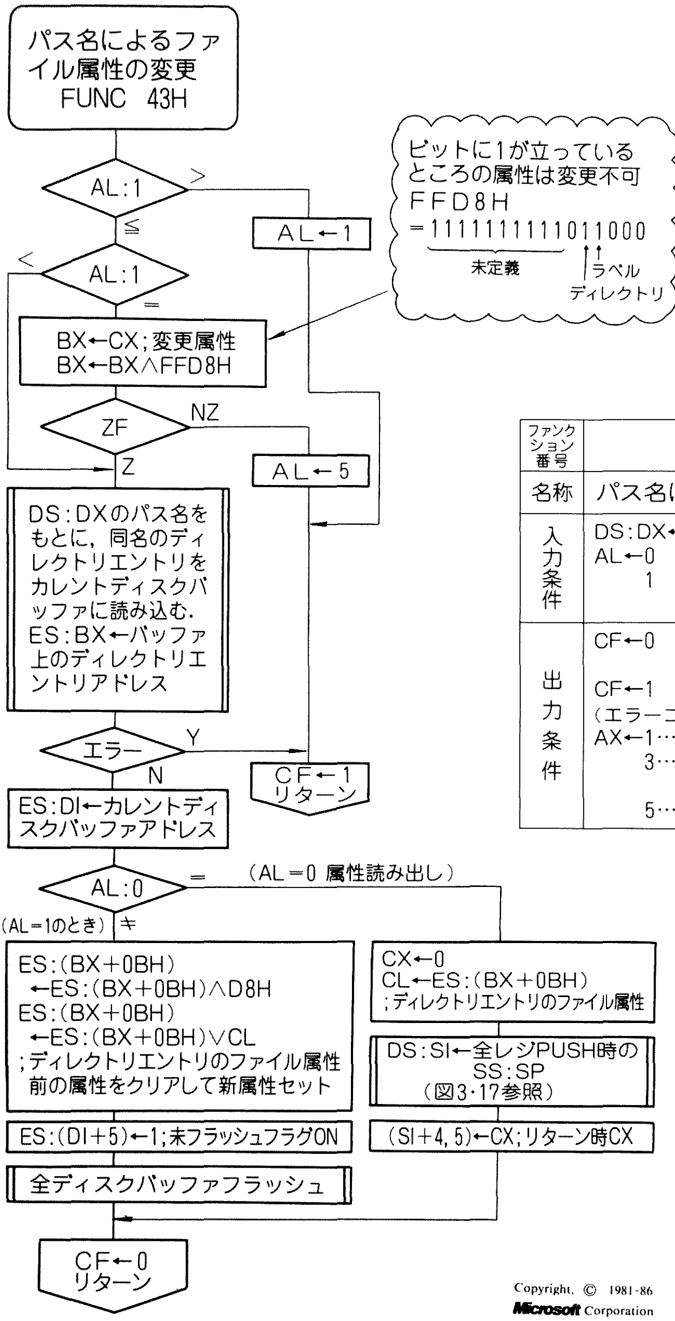
bit4　　　　ディレクトリ

bit6～bit15　未定義

があります．

処理としては，DS：DXの示すパスのディレクトリをカレントディスクバッファに読み込み，パス名のファイル名部と同名のディレクトリエントリをさがし，ALレコードに00Hがセットされていたらそのディレクトリエントリから属性を読み出し，ALレジスタに1がセットされていればCLレジスタで指定された属性を書き込みます．ファイル属性はディレクトリエントリの（+0BH）に格納されています．ディレクトリエントリのファイル属性格納エリアは1バイトしかとってありませんので，読み出すときは，上位バイトにはかならず00Hがセットされます．

図3・107に入出力条件と解析フローチャートを示します．



ファイル属性
1…リードオンリ
2…不可視
4…システムファイル
8…ボリュームラベル
10H…ディレクトリ
20H…保存ビット

ファンクション番号	43H
名称	パス名によるファイル属性の変更
入力条件	DS:DX←パス名アドレス AL←0　属性読み出し 　　1　属性セット 　　　CX…ファイル属性
出力条件	CF←0　正常終了 　　　CX…ファイル属性（読み出しのとき） CF←1　異常終了 （エラーコード） AX←1…与えられたALが0, 1以外 　　3…パス名としておかしい 　　　パスが存在しない 　　5…属性をディレクトリorラベルに変更しようとした

図 3・107　ファンクション 43H　パス名によるファイル属性変更の入出力条件と解析フローチャート

ファンクション44H
I/Oコントロール

INT 21H ファンクションコールから直接デバイスドライバにアクセスするために作られた，特殊なファンクションです．デバイスドライバはコマンド 0～12 の 13 種類の I/O リクエストコールというコマンドを持っていますが，ファンクション 44H でアクセスできるのは

I/O リクエストコール コマンド番号		I/O コントロール コマンド番号
4	リードデータ	2, 4
5	キュー非破壊入力	6
8	ライトデータ	3, 5
10	出力用ステータスチェック	7

だけです．

フローチャートでは，AL レジスタにセットしているコマンド番号が I/O リクエストコールコマンド番号と違っているようですが，これはブロック型デバイスに対しても”キュー非破壊入力”と”出力用ステータスチェック”（両者は本来 I/O リクエストコールではブロック型デバイスに対してはサポートされていない，**図 2・7** の”I/O リクエストコールコマンド一覧”参照）を使用できるように拡張した”I/O リクエストコール用サブルーチン（**図 3・31（a）（b）**参照）”用のコマンド番号だからです．

ファンクション 44H はそれ自体がさらにもう 1 つのファンクションといってよいほどのサブコマンドを持ち，そのサブコマンドは AL レジスタで指定することができます，各コマンドにどういうものがあるかは**図 3・108** を参照してください．

それでは各コマンドについて説明します．コマンド 0～3, 6, 7 は，ファイルハンドルの値から内部 FCB アドレスを求めます．

①コマンド 0…… デバイス属性の読み出し

キャラクタ型デバイスのときデバイスヘッダ（+5）を上位バイト，（FCB+18H）を下位バイトとして，ブロック型のときは上位バイトは 00H，下位バイトは（FCB+18H）すなわちドライブ番号（0 → A，1 → B……）を DX レジスタにリターンします．

DX レジスタにリターンされるそれぞれのビットの意味は

bit0	標準入力	下位 8 ビット
bit1	標準出力	
bit2	NUL デバイス	
bit3	CLOCK デバイス	
bit4	特殊デバイス	

bit5, 6	Don't Care	
bit7	キャラクタ型デバイス	
bit8～12	Don't Care	
bit13	IBM-PC のディスクフォーマット (MS-DOS 標準のディスクフォーマット）と異なる	上位 8 ビット
bit14	I/O コントロール使用不可	
bit15	キャラクタ型デバイス	

です.

②コマンド 1…… デバイス属性のセット

DX レジスタにデバイス属性をセットしますが，DH レジスタはかならず 0H にしておかねばなりません．さもなければエラー 0DH となります．下位 8 ビットの DL レジスタの値はそのまま (FCB＋18H) デバイス情報に書き込まれます．また，キャラクタ型デバイスでなければデバイス情報をセットすることはできません（エラー 1 となる).

③コマンド 2, 3…… キャラクタ型データリード/ライト

FCB の＋1C～1FH からデバイスヘッダアドレスを得て，デバイスヘッダ(＋4, 5)のデバイス情報の bit14 が 1 であることを確かめます．このビットは I/O コントロールをサポートしているかどうかを表わすビットです．あとは I/O リクエストコールのコマンド 4, 8 を実行するだけです.

このコマンドパケットを作るとき，アクセスするバイト数に CX レジスタの値，バッファアドレスに DS：DX の値を使います.

実際アクセスしたバイト数を AX レジスタにリターンします.

④コマンド 4, 5…… ブロック型データリード/ライト

BL レジスタで与えられたドライブの DPB アドレスを，ファンクション 32H を CALL して取得します．つぎに (DPB＋12H～15H) のデバイスヘッダアドレスを読み取ったら，あとはコマンド 2, 3 のキャラクタ型のリード/ライトと同じです.

このように，キャラクタ型とブロック型を分けたのは，ひとつにはキャラクタ型とブロック型で FCB の構造がすこし異なることも原因しています(**図 3•7** 参照)．すなわち，キャラクタ型は (FCB＋1C～1FH) にデバイスヘッダアドレスを持っていますが，ブロック型は DPB 内にそれがあり，ドライブ番号から指定しなければならないのです.

⑤コマンド 6…… 入力ステータスチェック

I/O リクエストコールコマンド 6 に"入力用ステータスチェック"というのがありますが，それを使わずに，I/O リクエストコールコマンド 5 の"キュー非破壊入力"を使用して入力状態のチェックをしています．AL レジスタに Ready のとき FFH, Busy のとき 00H をリターンしますが，Ready の意味は"入力タイプアヘッドバッファに情報が入っている"，Busy の意味は"入力タイプアヘッドバッファは空である"ということになります.

非破壊入力で得たコードは AH レジスタにリターンされます.

⑥コマンド 7…… 出力ステータスチェック

I/O リクエストコールコマンド 10 番"出力用ステータスチェック"を実行するだけです.
リターン情報は

AL : FFH Ready
　　 00H Busy

です. **図 3・108, 109** に入力と解析フローチャートを示します.

ファンクション番号	44H								
名称	I/Oコントロール								
各コマンドの入出力条件に従う	AL←コマンド番号								
	コマンド番号	0	1	2	3	4	5	6	7
	機能	デバイス情報リード	デバイス情報ライト	キャラクタ型データリード	キャラクタ型データライト	ブロック型データリード	ブロック型データライト	入力ステータスチェック	出力ステータスチェック
	入力条件	BX←ファイルハンドル	BX←ファイルハンドル	BX←ファイルハンドル	BX←ファイルハンドル	BL←ドライブ番号（0…カレントドライブ）	BL←ドライブ番号（0…カレントドライブ）	BX←ファイルハンドル	BX←ファイルハンドル
			DX←デバイス情報	CX←リード／ライトするバイト数 DS:DX←リード／ライトバッファ先頭アドレス	CX←リード／ライトするバイト数 DS:DX←リード／ライトバッファ先頭アドレス	CX←リード／ライトするバイト数 DS:DX←リード／ライトバッファ先頭アドレス	CX←リード／ライトするバイト数 DS:DX←リード／ライトバッファ先頭アドレス		
	出力条件	CF←0……正常終了							
		DX←デバイス情報	AL←変更前のデバイス情報	AX←実際アクセスされたバイト数	AX←実際アクセスされたバイト数	AX←実際アクセスされたバイト数	AX←実際アクセスされたバイト数	AL←FFH…Ready 00H…Busy AH←非破壊入力で得たコード	AL←FFH…Ready 00H…Busy
		CF←1……異常終了 AX←(エラーコード)							
		6…指定されたファイルハンドルは現在オープンされていない	6…指定されたファイルハンドルは現在オープンされていない 1…キャラクタ型でないデバイスを指定した 0DH…指定されたデバイス属性は無効	6…指定されたファイルハンドルは現在オープンされていない 1…キャラクタ型でないデバイスを指定した 1…またはデバイスがI/Oコントロールをサポートしていない	6…指定されたファイルハンドルは現在オープンされていない 1…キャラクタ型でないデバイスを指定した 1…またはデバイスがI/Oコントロールをサポートしていない	1…デバイスがI/Oコントロールをサポートしていない 5…ドライブ番号が最大ドライブ数を越えている	1…デバイスがI/Oコントロールをサポートしていない 5…ドライブ番号が最大ドライブ数を越えている	5…指定されたファイルハンドルは範囲外 指定されたファイルハンドルは現在オープンされていない	5…指定されたファイルハンドルは範囲外 指定されたファイルハンドルは現在オープンされていない
		1…コマンド番号が0〜7以外							

図 3・108 ファンクション **44H** **I/O** コントロールの入出力条件

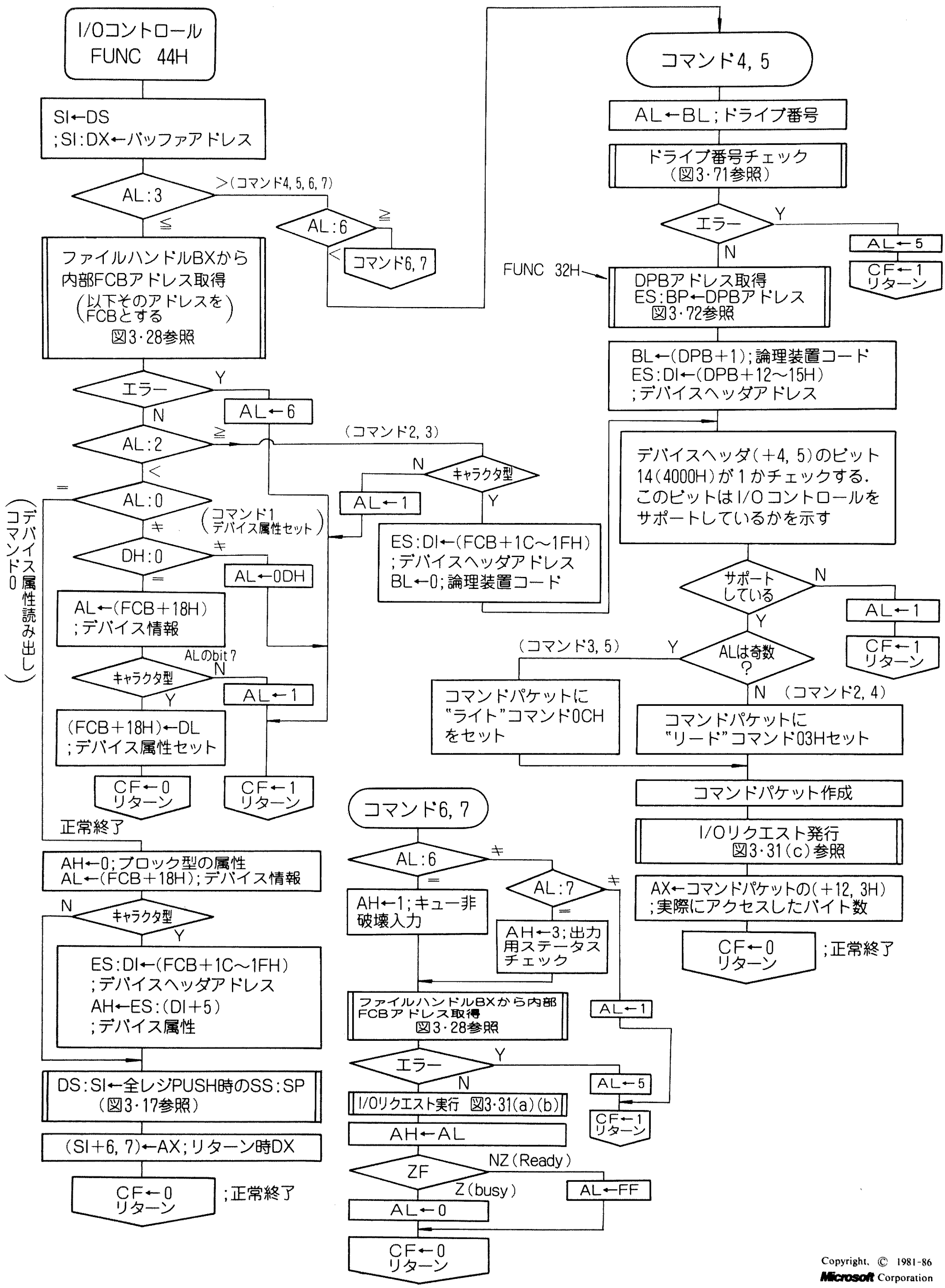


図3・109 ファンクション44H I/Oコントロールの解析フローチャート

ファンクション45H
ファイルハンドルのコピー

ファンクション46H
指定したファイルハンドルへのコピー

ファンクション 45H のほうは，空ファイルハンドルをさがし出してきて，そこへ BX で与えられたファイルハンドルをコピーするもので，ファンクション 46H のほうは，CX で与えられたファイルハンドルをファンクション 3EH”ファイルハンドルによるクローズ”を利用してクローズし，クローズによって空いたファイルハンドルへ BX で与えられたファイルハンドルをコピーするものです．

ファイルハンドルのコピーの例を**図 3・110** に示します．**図 3・110（a）**のような状態のときにファイルハンドル 2 をファイルハンドル 4 へコピーすると，**図 3・110（b）**のようになります．ファイルハンドル ⟷ 内部 FCB 番号対応表は PSP の＋18H〜2BH にあり，それぞれファイルハンドル 0

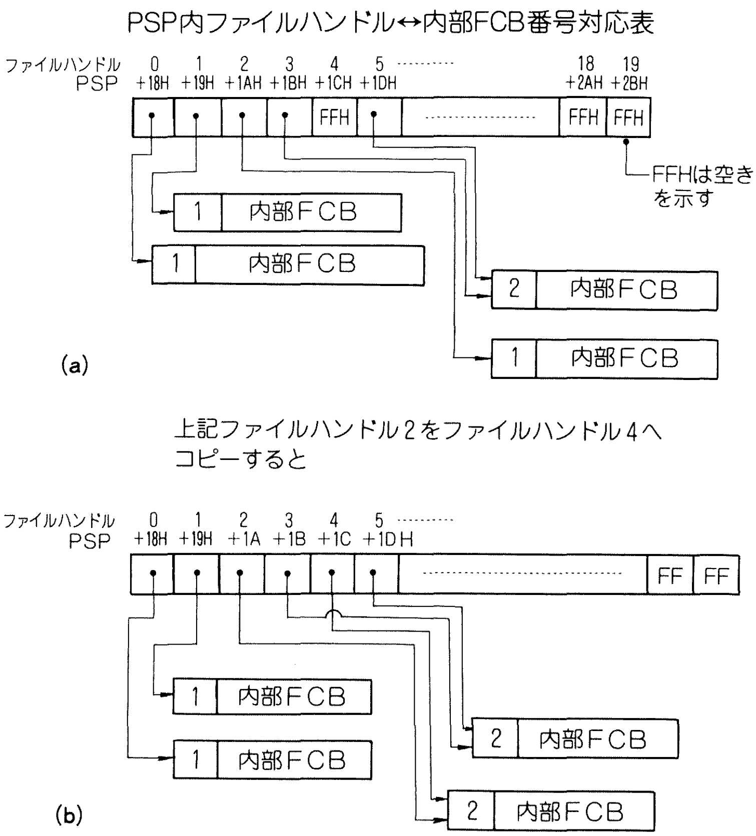


図 3・110　ファイルハンドルのコピー概念図

～19 に対応する内部 FCB 番号が格納されています．未使用のファイルハンドルには FFH が格納されています．ファンクション 45H では自動的にこの未使用ファイルハンドルを見つけ出してきて図のようにコピーするのです．もし未使用ファイルハンドルがなければ，エラー 6 となります．**図 3･110** で内部 FCB の先頭についている番号は，いくつのファイルハンドルから参照されているかを示します．

ただし，ここで PSP といっているのはカレント PSP です．実際には**図 3・111** のように各プロセスに PSP は存在し，その中にファイルハンドル←→内部 FCB 対応表は存在します．**図 3･111** でいうならば，カレント PSP には外部コマンドの PSP がセットされていますので，外部コマンドの PSP 内の対応表が変更されます（COMMAND . COM の PSP 内の対応表は変更されない）．したがって外部コマンドを終了したら，外部コマンド上での対応表は消えますが，COMMAND. COM の PSP 内の対応表は外部コマンド実行前と変わりません．

図 3・112 に入出力条件と解析フローチャートを示します．

COMMAND.COMのPSPの対応表

ファイルハンドル	0	1	2	3	4	5	6	7 ……
	1	1	1	0	2	FF	FF	…………

外部コマンドのPSPの対応表

ファイルハンドル	0	1	2	3	4	5	6	……
	1	1	1	0	2	FF	FF	…………

内部FCB番号

0	2	AUX
1	6	CON
2	2	PRN

図 3・111　実際のファイルハンドル←→内部 FCB 対応

ファンクション番号	45H	46H
名称	ファイルハンドルのコピー	指定したファイルハンドルへのコピー
入力条件	BX←コピー元ファイルハンドル	
		CX←コピー先ファイルハンドル
出力条件	CF←0 正常終了 AX…コピー先ファイルハンドル CF←1 異常終了 (エラーコード) AX←4…PSP内のファイルハンドル内部FCB対応表に空きがない オープンされているファイルハンドルが多すぎる (ファンクション45Hのみ) 6…指定されたファイルハンドルは現在オープンされていない	

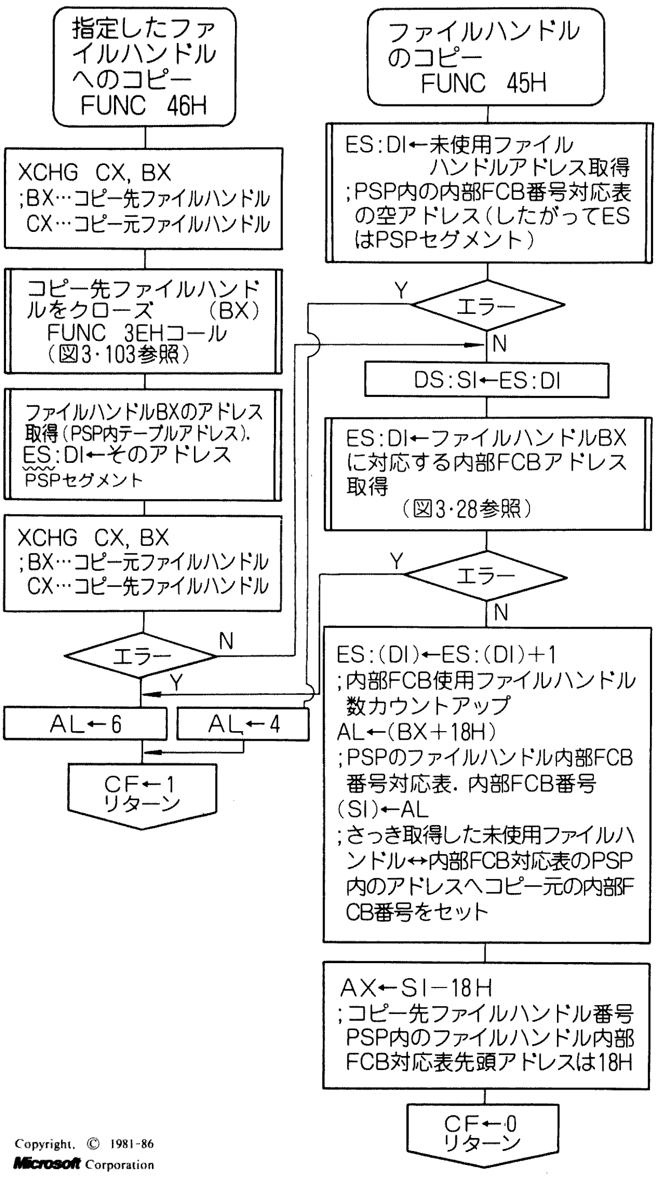


図 3・112 ファンクション 45H ファイルハンドルのコピー
ファンクション 46H 指定したファイルハンドルへのコピーの入出力条件と解析フローチャート

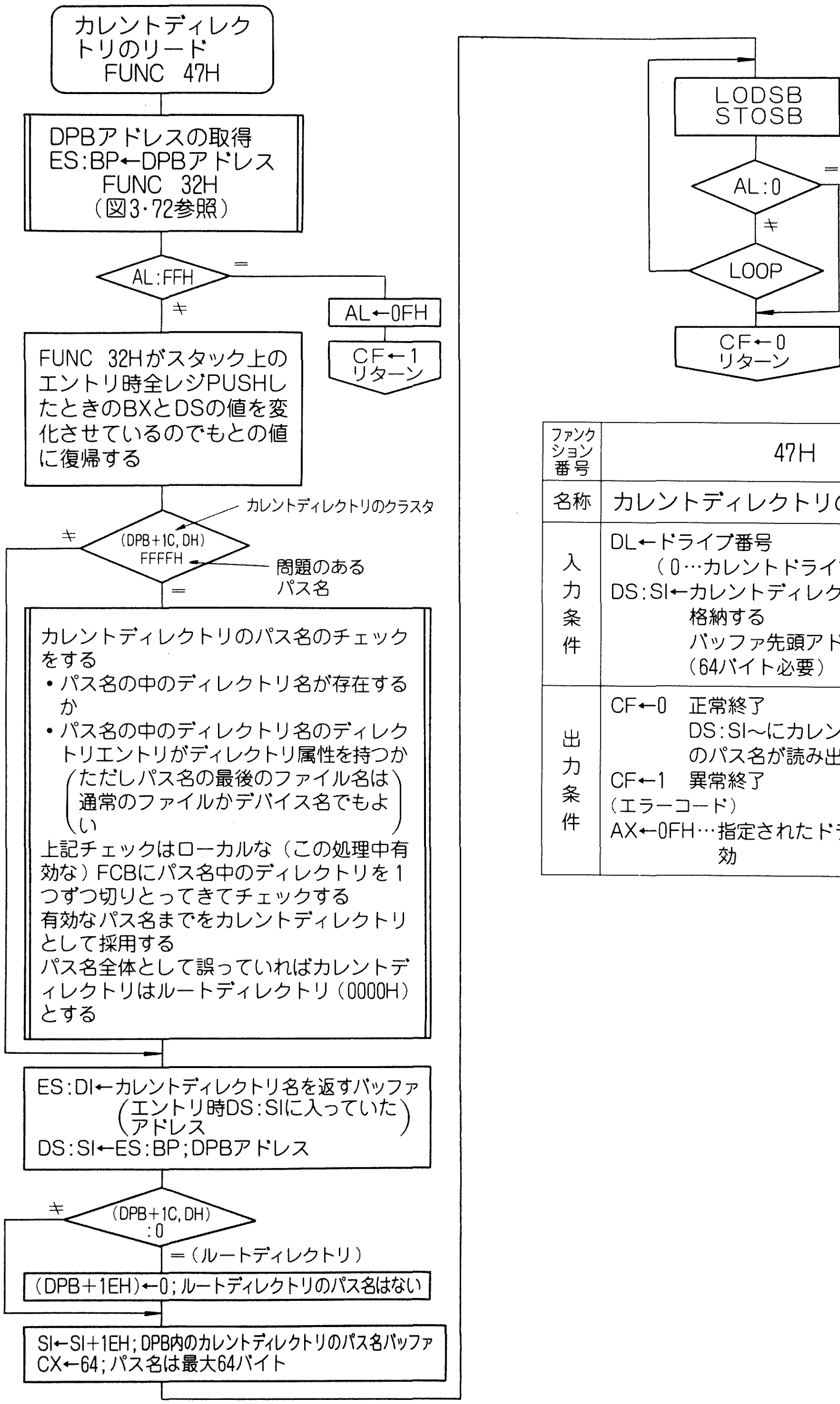
ファンクション47H
カレントディレクトリのリード

ブロック型デバイスでは各ドライブに対応してDPBが対応づけられており，DPBの(+1C, DH)にカレントディレクトリのクラスタ値，(+1E〜5DH)にカレントディレクトリのパス名が格納されています．処理としてはこれらの値を読み出すだけです．ただし，上記のカレントディクトリのパス名の格納エリアはバージョン2.xxのものであって，バージョン3.xxではシステムパラメータテーブル（**図3・3，図3・4**参照）の（+16H〜19H）にカレントディレクトリだけを集めたテーブルへのアドレスベクタが格納されているようです．バージョン3.xxであっても考え方は同じです．

ファンクション32HでドライブDLからDPBアドレスを求めます(**図3・72**参照)．DPBアドレスが求められれば，+1EH〜5DHに格納されているカレントディレクトリへのパス名を指定された（DS：SIで指定された）バッファへ転送します．

このとき，(DPB+1C，DH)のカレントディレクトリのクラスタが0000Hのときは，ルートディレクトリなのでカレントディレクトリ名は00H(ヌルストリング)で名前なしとなり，FFFFHのときはカレントディレクトリの変更（ファンクション3BH）のときにカレントディレクトリパス名が64バイトを越えてしまったということですから，再度(DPB+1E〜5DH)のパス名のチェックを行います．そしてパス名として有効な部分までをカレントディレクトリとし，ディレクトリ名として成り立たない文字列であれば，カレントディレクトリはルートディレクトリとしています．

図3・113に入出力条件と解析フローチャートを示します．



ファンクション番号	47H
名称	カレントディレクトリのリード
入力条件	DL←ドライブ番号 (0…カレントドライブ) DS:SI←カレントディレクトリのパス名を格納するバッファ先頭アドレス (64バイト必要)
出力条件	CF←0 正常終了 DS:SI〜にカレントディレクトリのパス名が読み出された CF←1 異常終了 (エラーコード) AX←0FH…指定されたドライブ番号が無効

図3・113 ファンクション47H カレントディレクトリのリードの入出力条件と解析フローチャート

ファンクション48H
MCB割り当て

BX レジスタで要求されたパラグラフ数のメモリブロックを，フリーメモリから確保するだけです．確保に成功したら割り当てられたMCBのセグメントをAXにリターンします．

処理としては，前半と後半に（**図3・115**，**116**のフローチャートで（A）を境にして）分けられ，前半では先頭のMCBからリンクをたどって最終のMCBまでサーチする間に3種類のMCBを求めます．1つめは最初の無所属MCBセグメント（すなわち，最下位アドレスに位置するフリーなMCB)．2つめは最小サイズの無所属MCBセグメント（MCBの（+3，4）”MCBのサイズ”のいちばん小さいフリーなMCB)，3つめは最後の無所属MCBセグメント(すなわち，最上位アドレスに位置するフリーなMCB)を求めるのです．ここで”無所属”といっているのはMCBの(+1，2)”MCBの属するパラグラフアドレス”が0000Hのもののことです．ふつうMCBの(+1，2)にはそのMCBの属するメモリが管理されているPSPの先頭セグメントアドレスが入れられていますが，ここが0000Hということは，このメモリブロックを管理するPSPがない…すなわち未使用メモリブロック

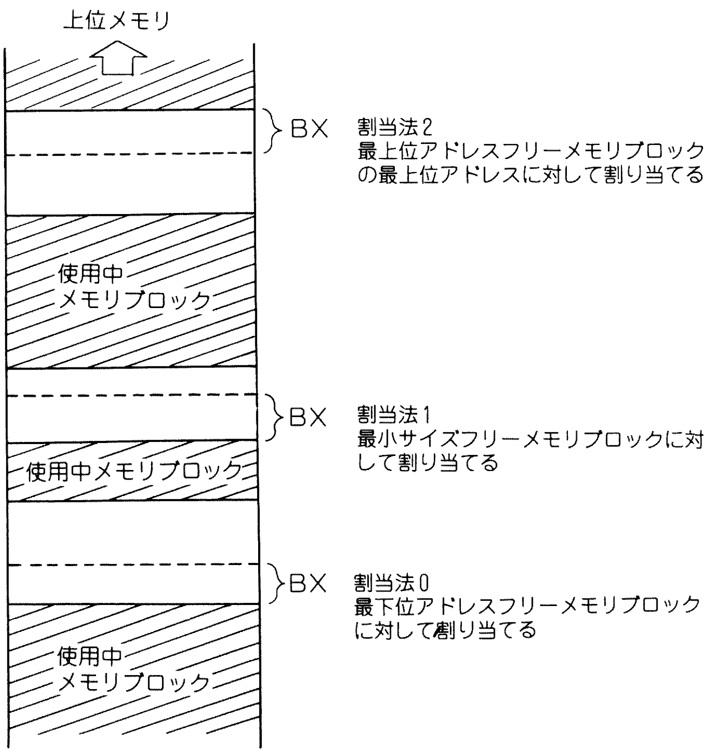


図3・114　MCB割り当て法の違い

ファンクション番号	48H
名称	MCB割り当て
入力条件	BX←割り当て要求パラグラフサイズ
出力条件	CF←0　正常終了 　AX…割り当てられたセグメント CF←1　異常終了 　BX…割り当て可能なパラグラフ数(エラー8のとき) (エラーコード) AX…7　MCBが壊れている(ブロック属性にM, Z以外のものがある) 　8　割り当てに十分なメモリがない

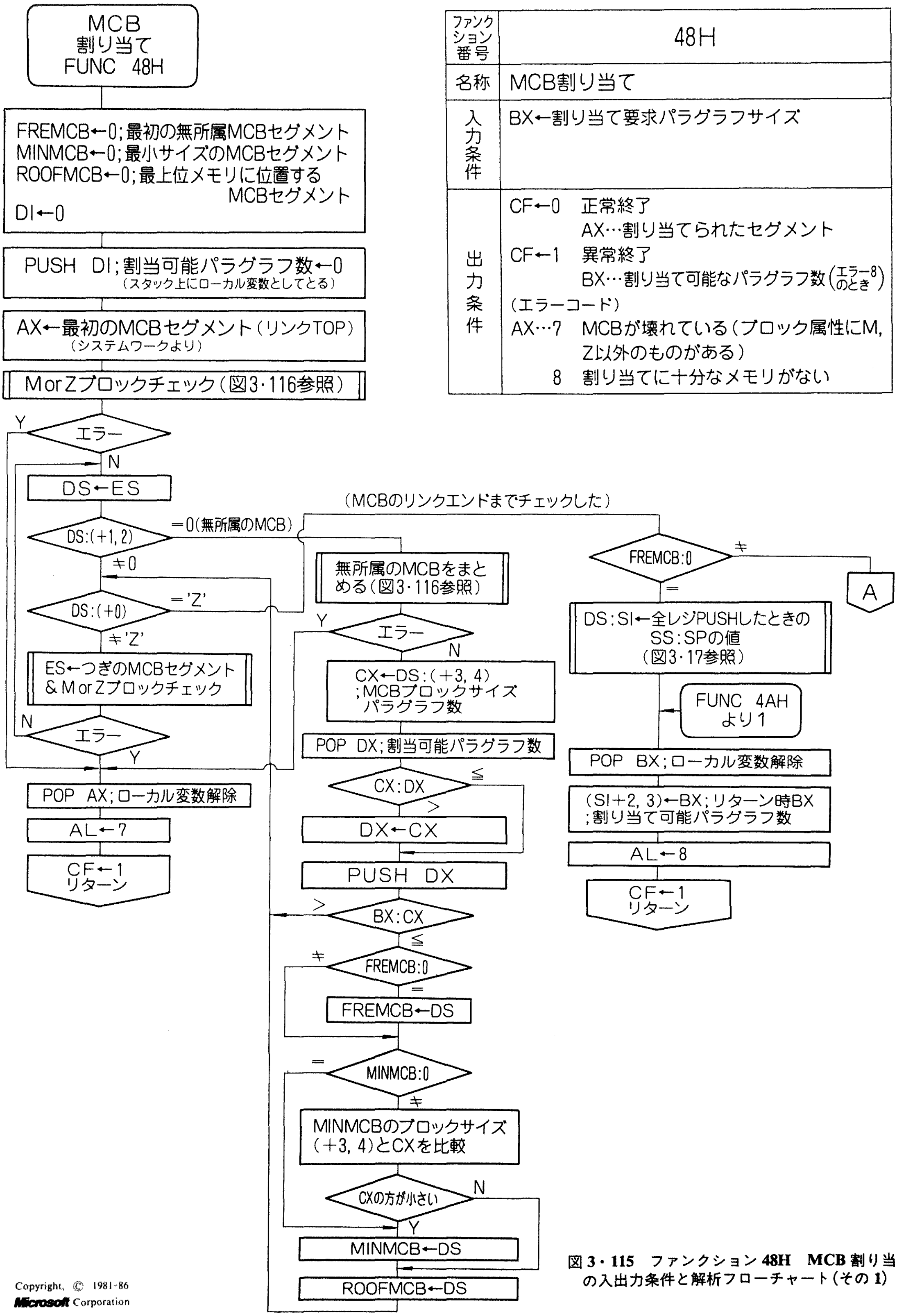


図3・115　ファンクション48H　MCB割り当ての入出力条件と解析フローチャート(その1)

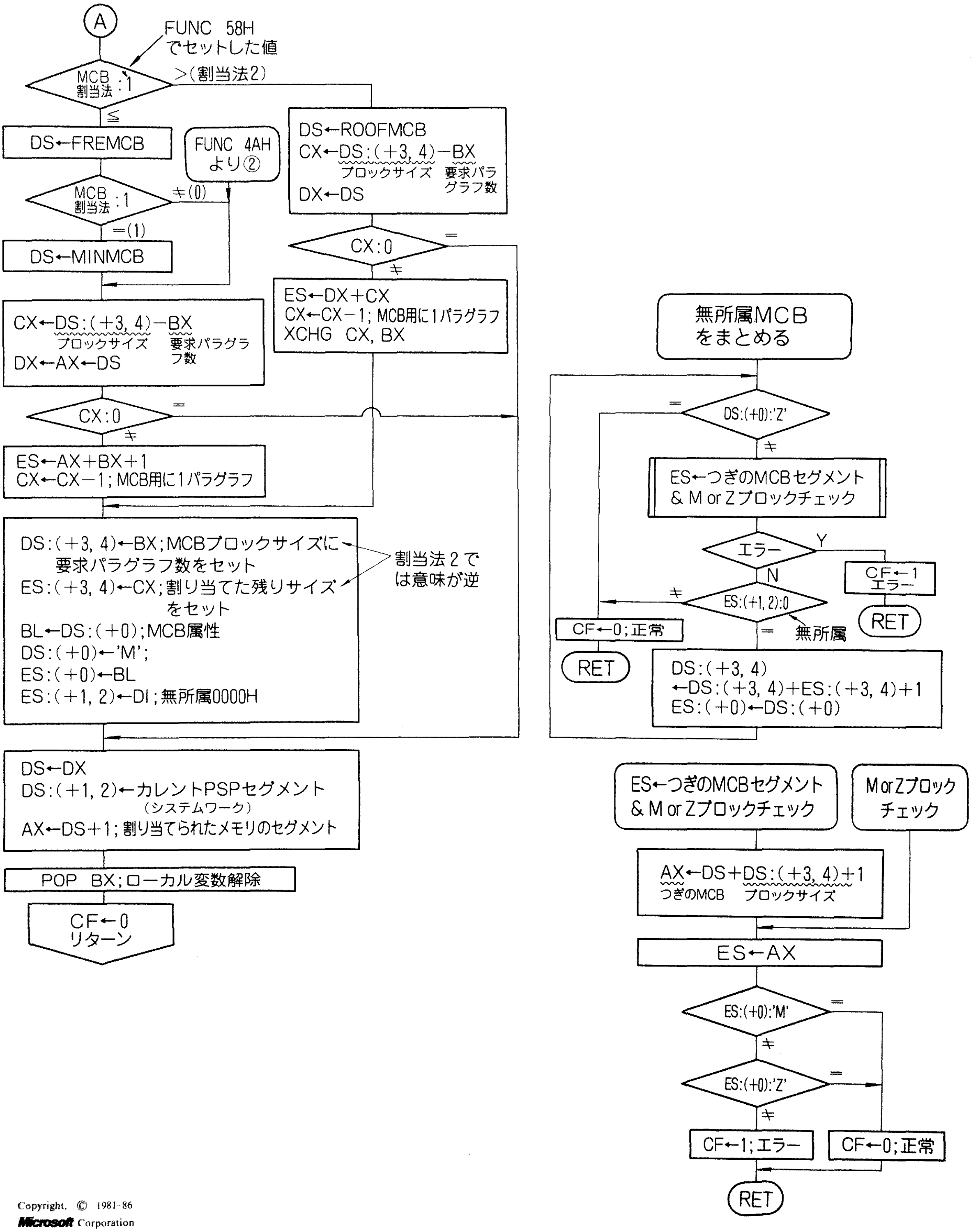


図 3・116 ファンクション 48H MCB 割り当ての解析フローチャート(その 2)

であることを表わします.

そして後半の処理にゆくのですが, ここで, 最初の無所属MCBが見つからなかった(FREMCB=0)ということは, 割り当て要求パラグラフより大きい未使用メモリがなかったということでエラー8(割り当てに十分なメモリがない)ということになります. ここでBXレジスタに割り当て可能パラグラフ数がリターンされますが, これは最大サイズの無所属MCBのパラグラフサイズです.

さて後半の処理ですが, 前半まで求めた3種のMCBを使って, MCB割り当て法フラグ(ファンクション58Hでセットされるフラグ, デフォールトは0)にしたがってMCBを割り当てます.

MCB割り当て法にも3種類あって,

0　最小アドレスに割り当てる

1　最小サイズのメモリブロックに割り当てる

2　最上位アドレスに位置するメモリブロックに割り当てる

があります. それぞれ, 割り当て法0に対しては最初の無所属MCB(FREMCB), 割り当て法1に対しては最小サイズの無所属MCB(MINMCB), 割り当て法2に対しては最上位メモリのMCB(ROOFMCB)をもとに割り当て法が実行されます.

図3•114に上記割り当て法による違いの概念図を示します. また, 前半の処理の中で連続する無所属のMCBを見つけたら, 1つのMCBにまとめます. MCBの先頭が'M'か'Z'でなければメモリブロックが破壊されているとみなします.

図3・115, 116に入出力条件と解析フローチャートを示します.

ファンクション49H
MCB解放

処理としては与えられたMCBの（+1, 2）の所属セグメントを0000Hにするだけです．所属セグメントが0000HになったMCBをまとめる処理は，MCBの割り当てやサイズ変更のほうで行います．

解放したいMCBはESレジスタで指定しますが，MCBのパラグラフ+1を指定します．MCBとメモリブロックは，**図3・117**のようにMCBの1パラグラフ上にメモリブロックがある関係になっているためです．

図3・118に入出力条件と解析フローチャートを示します．

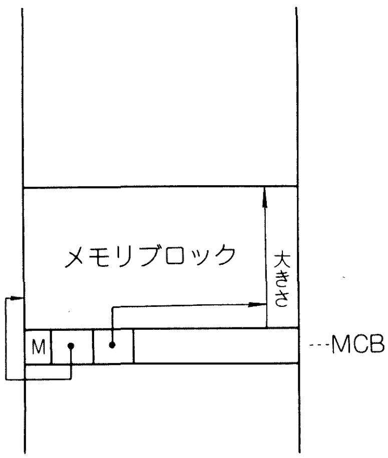
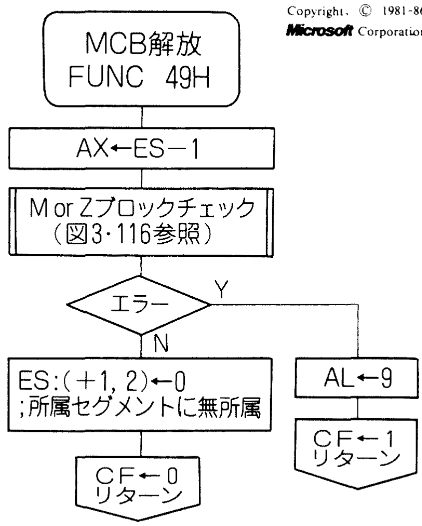


図3・117　メモリブロックとMCBの位置関係



ファンクション番号	49H
名称	MCB解放
入力条件	ES←解放するメモリセグメント
出力条件	CF←0　正常終了 1　異常終了 （エラーコード） AX…9　無効なメモリブロック （MCB属性がM, Z以外）

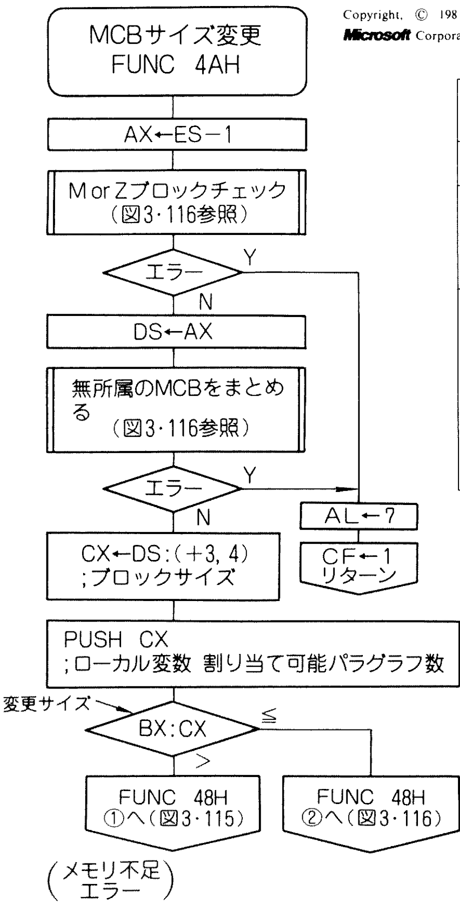
図3・118　ファンクション49H　MCB解放の入出力条件と解析フローチャート

ファンクション4AH
MCBサイズ変更

サイズ変更したいMCBをESレジスタで指定し，変更したいサイズをBXレジスタに入れて実行します．ここでESには変更したいMCBのセグメント＋1を入れておきます．理由はファンクション49Hと同じなので，**図3・117**を見てください．

処理としては，まずESレジスタで指定されたMCBからうしろのフリーMCBを1つにまとめます(**図3・116**参照)．その後変更サイズBXとMCBの(＋3, 4)のメモリブロックサイズを比較して，MCBのサイズのほうが小さければ，メモリ不足でBXレジスタにMCBサイズをリターンします．変更サイズのほうが小さければ変更可能で，ファンクション48Hの割り当て法チェックのうしろからエントリして，割り当てします．

図3・119に入出力条件と解析フローチャートを示します．



ファンクション番号	4AH
名称	MCBサイズ変更
入力条件	ES←サイズ変更したいMCBのセグメント BX←変更したいパラグラフサイズ
出力条件	CF←0　正常終了 CF←1　異常終了 BX…割り当て可能なパラグラフ数(エラー8のとき) (エラーコード) AX…7　MCBが壊れている(ブロック属性にM, Z以外のものがある) 8　変更に十分なメモリがない

図3・119　ファンクション4AH　MCBサイズ変更の入出力条件と解析フローチャート

ファンクション4BH
プログラムのロード&実行

ブロック型デバイス上に格納されている実行型ファイルを，メモリ上に読み出し実行するファンクションです．または，メモリ上に読み出すだけで実行はしない（ロードだけ）ということもできます（ロードするだけのほうも，ただロードするのと，スタックなどを用意してエントリすれば実行できるような形でロードするものとがある）．

ALレジスタに指定する機能コードによって

0 … ロード&実行

1 … ロードのみ．パラメータブロックにSS：SP，CS：IPの初期値を得る

2 … ロードのみ

の3種類があります．

実行型ファイルにはEXE型とCOM型があります．EXE型にはプログラム実行に必要な諸元を記録したヘッダがありますが，COM型にはヘッダはありません．EXE型のヘッダにはSS：SPやCS：IPの初期値，プログラム中セグメント値を使っている部分の位置などが記録されています．**図3・120**にEXE型ファイルヘッダの内容を示します．

オフセット	内　　　容
+0，1	4DH，5AH（MZ）または5AH，4DH（ZM）…EXEファイルID
+2，3	最終ページ内のバイト数
+4，5	EXEファイルの大きさ（ページ単位，1ページ…512バイト）
+6，7	リロケーション数
+8，9	ヘッダの大きさ（パラグラフ単位，1パラグラフ…16バイト）
+0A，BH	プログラムのうしろに必要な最小メモリ（パラグラフ単位）
+0C，DH	プログラムのうしろに必要な最大メモリ（パラグラフ単位） 0000Hのときメモリ最上位に読み込む
+0E～11H	SS：SPの初期値
+12，3H	チェックサム
+14～17H	CS：IPの初期値
+18，19H	リロケーションテーブルの始まりのファイルポインタ
+1A，1BH	オーバレイ番号
	リロケーション情報

図3・120　EXEファイルヘッダ

パラメータブロックフォーマット

AL←0 ロード&実行，AL←1 ロード&SS:SP，CS:IPの初期値リターン

+0，1	環境のセグメント	AL=0のとき	AL=1のとき
+2〜+5	デフォールトのDTA(PSPの+80H〜FFH)に与える文字列が格納されているバッファアドレス	AL=0のとき	AL=1のとき
+6〜+9	デフォールトのFCB(PSPの+5CH〜)に与えるデータバッファアドレス	AL=0のとき	AL=1のとき
+0A〜+0DH	デフォールトのFCB(PSPの+6CH〜)に与えるデータバッファアドレス	AL=0のとき	AL=1のとき
+0E，FH	SPの初期値−2 (スタックトップにはデフォールトのFCBのドライブ番号部分のチェック結果が入っている)	この値がリターンされる	AL=1のとき
+10，1H	SSの初期値	この値がリターンされる	AL=1のとき
+12，3H	IPの初期値	この値がリターンされる	AL=1のとき
+14，5H	CSの初期値	この値がリターンされる	AL=1のとき

AL←3 ロードのみ

+0，1	プログラムロードセグメント(注)
+2，3	再配置因子．プログラム中セグメントを使っているところにこの値が足される．

(注) ロードのみの場合ロード用メモリはプログラムをロードするために十分なメモリがユーザーによって確保されているものと仮定してロードされる

図3・121　パラメータブロックの内容

このファンクションの処理の大まかな流れは

① 環境作成（機能コード0，1のとき）
② プログラムロード用メモリ確保
③ プログラムロード
④ CS：IP，SS：SPの初期値計算
⑤ PSPの作成
⑥ 実行（CS：IPの初期値へジャンプ．機能コード0のとき）

となっています．EXE型とCOM型で②③④の処理内容が大きく異なりますので，この部分はEXE型とCOM型に分けて解説します．

まず最初は，ファイルをオープンするのにファンクション3DH（パス名によるオープン）を使っています．ここで得られたファイルハンドルを使って以降の処理は行われます．I/Oコントロール(ファンクション44H)のコマンド0(デバイス情報リード)によってデバイス情報を読み出し，bit7=1ならキャラクタ型ですからエラー2で落とします．実行するプログラムはブロック型デバイスからしか読み込めないようになっています．

機能コード3（読み込みだけ）以外（機能コード0，1）のとき環境を作成します．環境は新しくメモリを確保し，パラメータブロックアドレス(ファンクション4BHエントリ時ES：BXで指定していたテーブルアドレスのことで**図3・121**のようになっている）の（+0，1）で指定されている環境の内容をコピーして作成するだけです．この（0，1）の値が0000Hのときは新しく環境を作成せ

ず，カレントPSPの（+2C，DH）に格納されている環境セグメントアドレスをそのまま使用します．すなわち，カレントPSPの環境をそのまま使用することになります．

つぎに，MSDOS.SYSのシステムのワークエリア内に確保した1EHバイトのバッファに，さきほどパス名でオープンしたファイルの先頭1EHバイトだけ読み込みます(ファイルハンドルによるリード，ファンクション3FH利用)．

以上はEXE型，COM型共通の処理ですが，ここでEXE型とCOM型で処理が分かれます．

COM型の判断は

① 実際に読み込まれたバイト数が1EHと異なる

② 1EHバイト読み込んだ0バイト目と1バイト目が'MZ'か'ZM'でない

のときCOM型のプログラムロード，CS：IP・SS：SPの初期値計算処理へ飛びます．上記以外，すなわちプログラム先頭の1EHバイトが読み込めて，先頭2バイトが'MZ'か'ZM'のときはEXE型の処理に進みます．

(EXE型の処理)

●プログラムサイズの計算

プログラムサイズは以下の式によって求めます．

(EXEヘッダ+4，5) ＊ 32－ (EXEヘッダ+8，9) [パラグラフ]

EXEファイルの大きさ　　　　リロケータブル情報のサイズ

EXEファイルの大きさはページ単位なので32倍してパラグラフ単位にしています．

ここで機能コード3（ロードのみ）であればつぎの処理は行いません．

●ロードするメモリの確保

MCB割り当て要求に対してFFFFHパラグラフを指定し，割り当て可能最大パラグラフ数を求めます．さきほど計算したプログラムサイズとつぎの関係があることをチェックします．

(割り当て可能最大パラグラフ数)＞＝MIN（プログラムサイズ+10H，11H)

プログラムサイズに10Hを足しているのは，PSPのパラグラフ数です．満足していなければメモリ不足エラーで落とします．

EXEのヘッダ+0C，DH"プログラムのうしろに必要な最大メモリ"が0000Hでなければ，つぎのチェックをします．

(プログラムサイズ+10H)＋（EXEヘッダ+0A，BH)

プログラムのうしろに必要な最小メモリ

＜＝MIN（割り当て可能最大パラグラフ数，FFFFH)

を満足していなければメモリ不足エラーで落とします．

(EXEヘッダ+0C，DH）のほうは

(プログラムサイズ+10H)＋（EXEヘッダ+0C，DH)

プログラムのうしろに必要な最大メモリ

＜＝MIN（割り当て可能最大パラグラフ数，FFFFH)

をチェックして，満足していれば

(プログラムサイズ+10H)＋（EXEヘッダ+0C，DH)

を，満足していなければ

(割り当て可能最大パラグラフ数)

を指定して再度MCB割り当てを実行します.割り当てが成功すれば,割り当てられたセグメントが決定されます.これがPSP用割り当てセグメントとなり,それに+10Hしたものがプログラム本体ロード用セグメントとなります.もし(EXEヘッダ+0C, DH)が0000Hのときはプログラムを最上位アドレスにロードすることになり,プログラム本体ロード用セグメント

(割り当てられたセグメント) + (割り当てられたパラグラフ数)
　　　　　　　　　　　　　　　－ (プログラム本体の大きさ)

となります.

●メモリへの読み込み

プログラム本体の大きさはパラグラフ単位で計算していますので,ファンクション3FHの"ファイルハンドルによるファイルリード"で読み込むにはバイト単位に換算しなければなりません.プログラム本体の大きさが64kバイトを越える場合は,FE00Hバイトずつの単位で分割して読み込みます.

● SS:SP, CS:IPの初期値の計算

SP, IPの初期値はそれぞれEXEヘッダ(+10H, 11H)と(+14H, 15H)の値をそのまま使用しますが,SS, CSの初期値はEXEヘッダ(+0E, FH)と(+16H, 17H)に(プログラム本体ロードセグメント)を再配置因子として足したものになります.

●リロケーションの処理

プログラムの中でセグメント値を使用するものは"プログラム本体ロードセグメント"を再配置因子として足さなければなりません(ただし,機能コード3のロードのみの場合は再配置因子はパラメータブロック(**図3・121**参照)の(+02, 3H)の値を使用します).

プログラム中のどこにセグメント値を使っているかは,EXEファイルヘッダのリロケーション情報の部分にそのセグメントを使っているオフセットとセグメント値としてプログラム中のセグメントを使っている位置を示しています.したがって,プログラム中の位置を示すセグメントに再配置因子を足してやって,メモリ中に読み込まれたプログラムの実際のアドレスに変換してやり,そこにセグメント値(メモリに読み込んですぐはプログラム先頭をセグメント0とした値になっている)があるはずですから,ふたたびそのアドレスが指すところへ再配置因子を足し込んで実際のセグメント値とするのです.

リロケーション情報は4バイトで一組ですが,上記処理をするのにMSDOS.SYS内にリロケーション情報バッファとして1CHバイトしかとっていないので,リロケーション情報7個ずつ読み出して処理することになり,EXEプログラムの実行はリロケーションの書き換えのオーバーヘッドのためもあって,COM型のロード時間より遅くなるのです.

ここまででプログラムは実行できる形でメモリ上へロードできているので,COM型の処理のうしろの共通後処理へ合流します.

(COM型の処理)

●ロードするメモリの確保

機能コード3(ロードだけ)ならパラメータブロック(**図3・121**参照)の(+0, 1)のプログラ

ムロードセグメントにロードすればよいのですが，機能コード 0, 1 の場合は MCB 割り当てに対して FFFFH を要求し割り当て可能最大パラグラフ数を求めます．その値を使って再度 MCB 割り当て要求をし，実際に割り当てられたセグメント値を得ます．それを PSP のセグメントとし，+10H したものをプログラム本体のロードセグメントとします．

● メモリへの読み込み

読み込みバイト数を以下の式によって計算します．

(読み込みバイト数)＝MIN（(割り当て可能最大パラグラフ数）＊ 32，FFFFH）

これだけのバイト数をファイル先頭から，プログラム本体ロードセグメントへ読み込みます．

● SS：SP，CS：IP の初期値の計算

CS の初期値　＝PSP 用割り当てセグメント
IP の初期値　＝100H
SS の初期値　＝PSP 用割り当てセグメント
SP の初期値　＝MIN（割り当て可能最大パラグラフ数＊ 32，FFFFH)－1

として，スタックトップに 0000H をセットします．プログラムが RET で終了しても PSP の 0000H 番地へ飛ぶのでプログラム終了とすることができるからです．すなわち，COM 型のプログラムはプログラム実行初期のスタックトップの状態であれば RET で終了するこもできるのです（ただし，Ver3．xx 以降の MS-DOS は RET では終了できなくなるでしょう）．

以上で COM 型ファイルはメモリへ読み込まれました．

(共通後処理)

ファイルハンドルによるファイルクローズ（ファンクション 3EH）を CALL して，ファイルをクローズします．

機能コード 3（ロードだけ）の場合は以上で終了です．

機能コード 0, 1 の場合は続けて PSP を作成します．PSP はカレント PSP をもとにしてファンクション 55H で作成します．環境の MCB と PSP の MCB の（+1，2）に PSP のセグメントを書き込みます．

デフォールトの FCB と DTA には，パラメータブロック（**図 3・121** 参照）で指定された文字列をコピーします．デフォールトの FCB のドライブ番号部はチェックして，OK であれば 00H，NG であれば FFH をリターンします．機能コード 1 であればパラメータブロックの+0EH～14H，15H に SS：SP，CS：IP の初期値等をリターンし，機能コード 0 であれば INT 21H 実行中フラグ（IN-DOS フラグ）を OFF にして，SS：SP の初期値は実際に SS：SP レジスタにセットし，デフォールトの FCB のドライブ番号チェック結果は AX レジスタに入れ，CS：IP の初期値で示されるアドレスにジャンプします．

図 3・122～126 に入出力条件と解析フローチャートを示します．

なお，本項にはよくパラグラフという言葉が出てきますが，パラグラフとは 16 バイト単位のことです．すなわち，1 パラグラフは 16 バイトで，したがって，メモリの大きさを指定するときにパラグラフというと，その数字を 16 倍したものがバイト数になります．16 進数字でいえば，最後に 0 をつければパラグラフから，バイトへの変換になります．

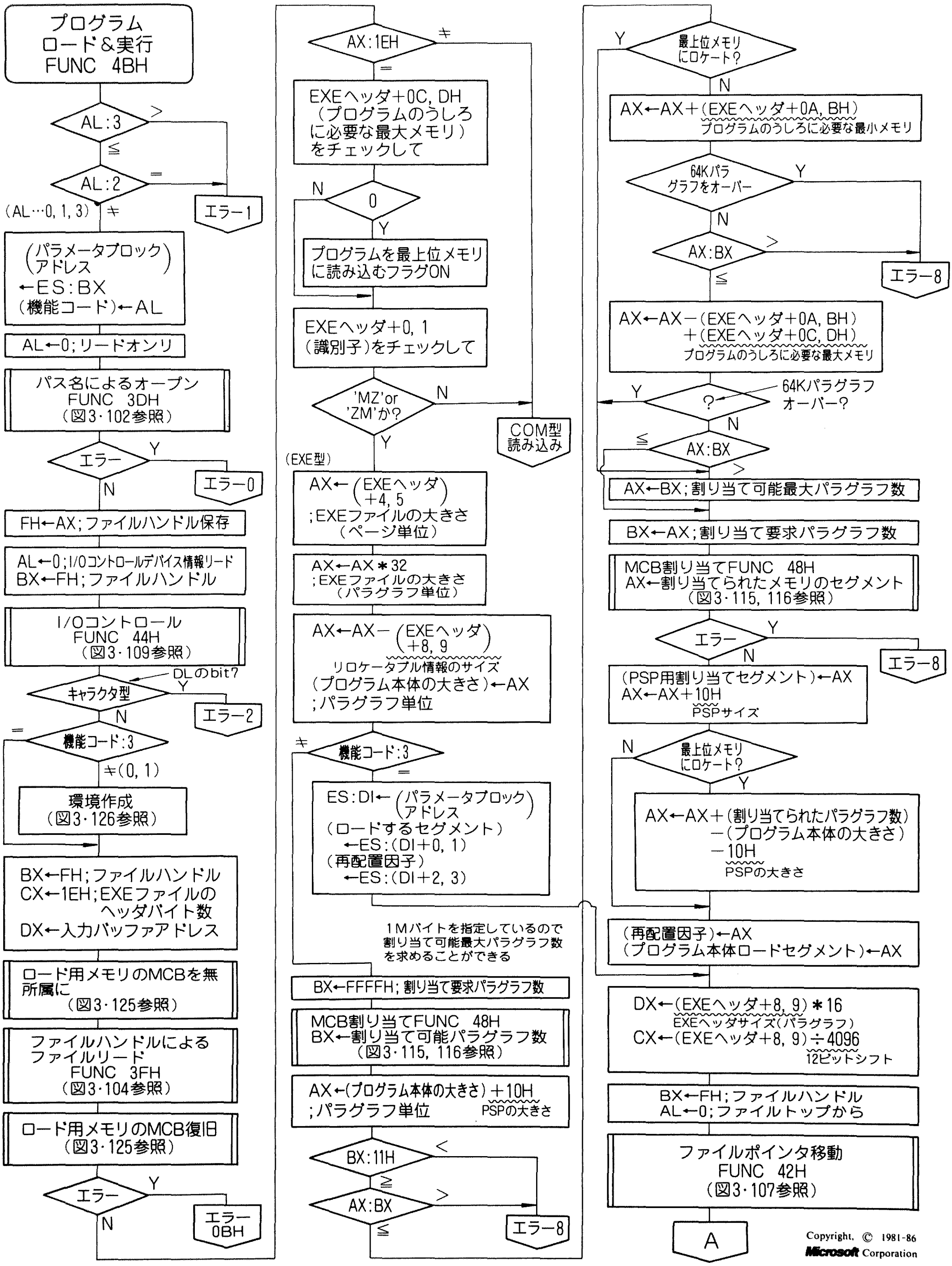


図3・122 ファンクション4BH プログラムのロード&実行 解析フローチャート その1

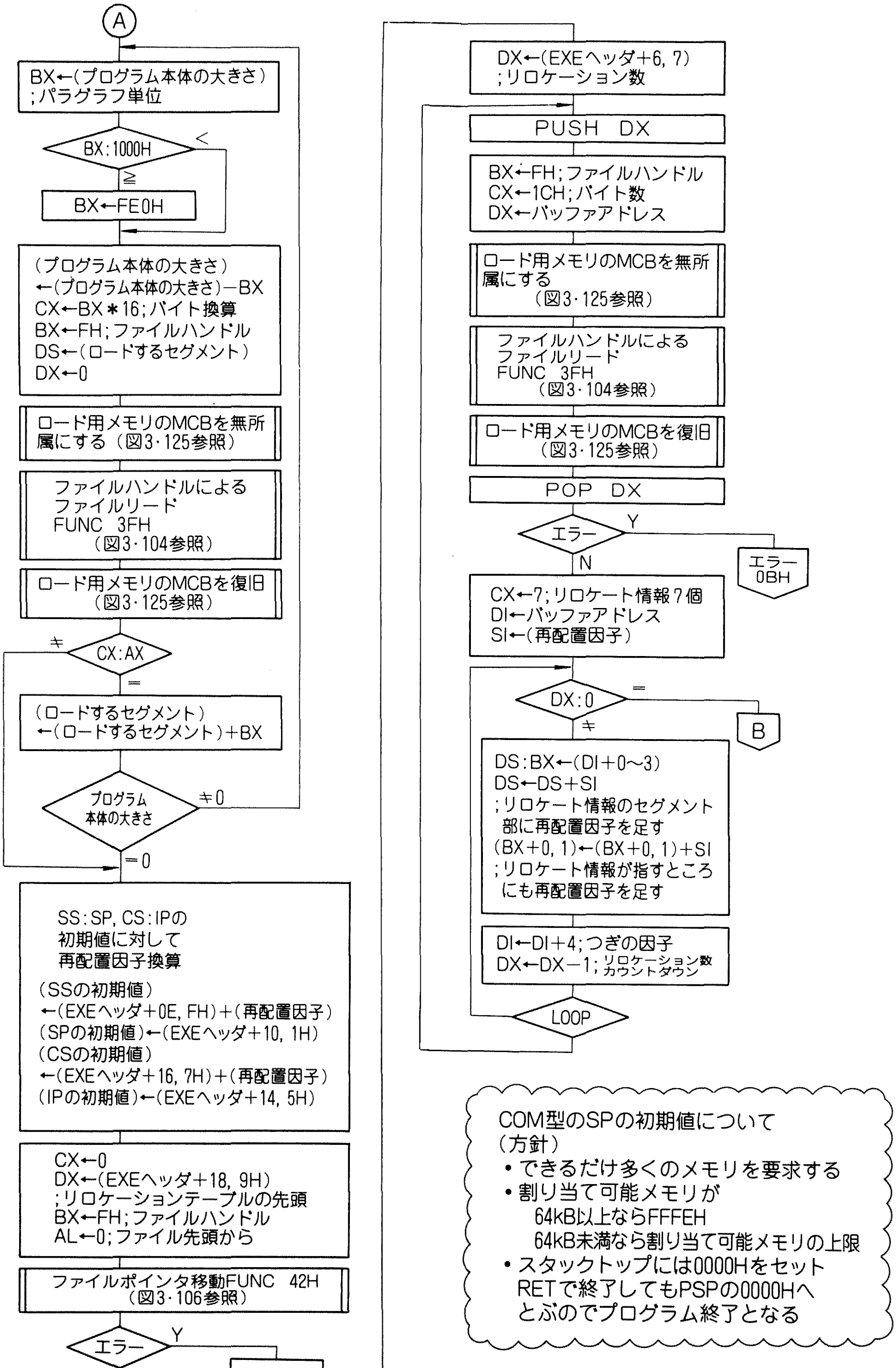


図3・123 ファンクション4BH プログラムのロード&実行 解析フローチャート その2

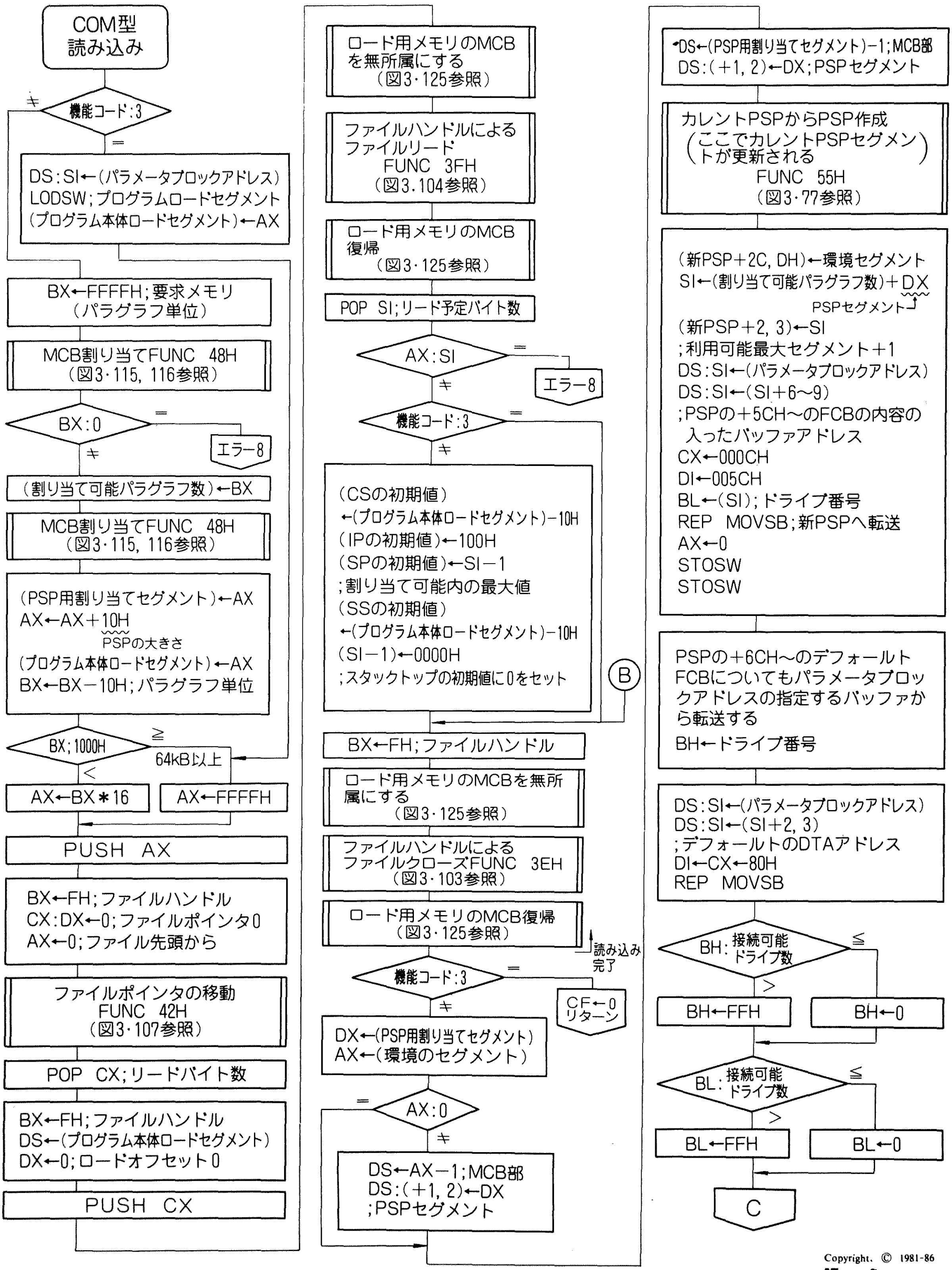


図 3・124 ファンクション 4BH プログラムのロード&実行 解析フローチャート その3

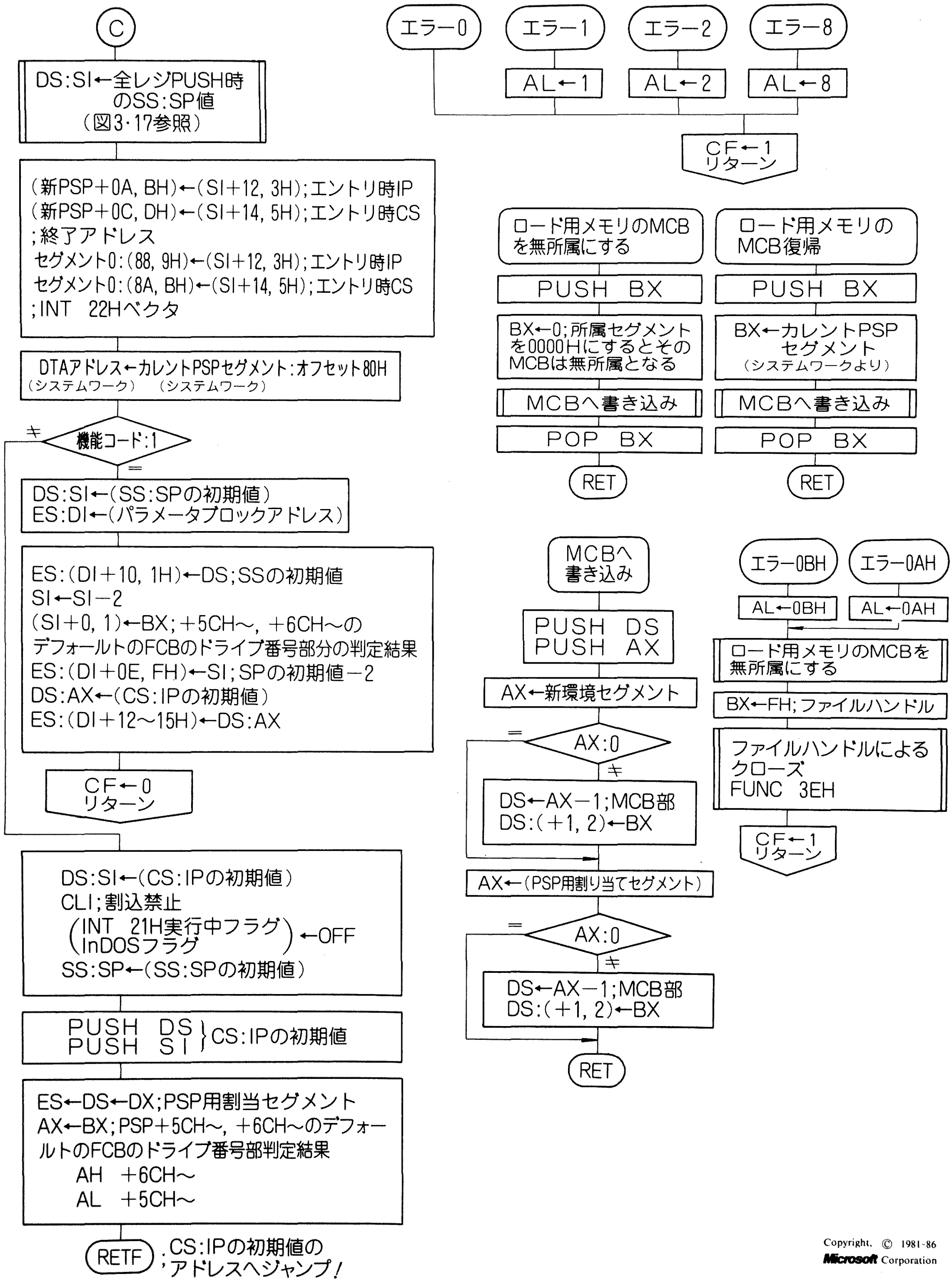
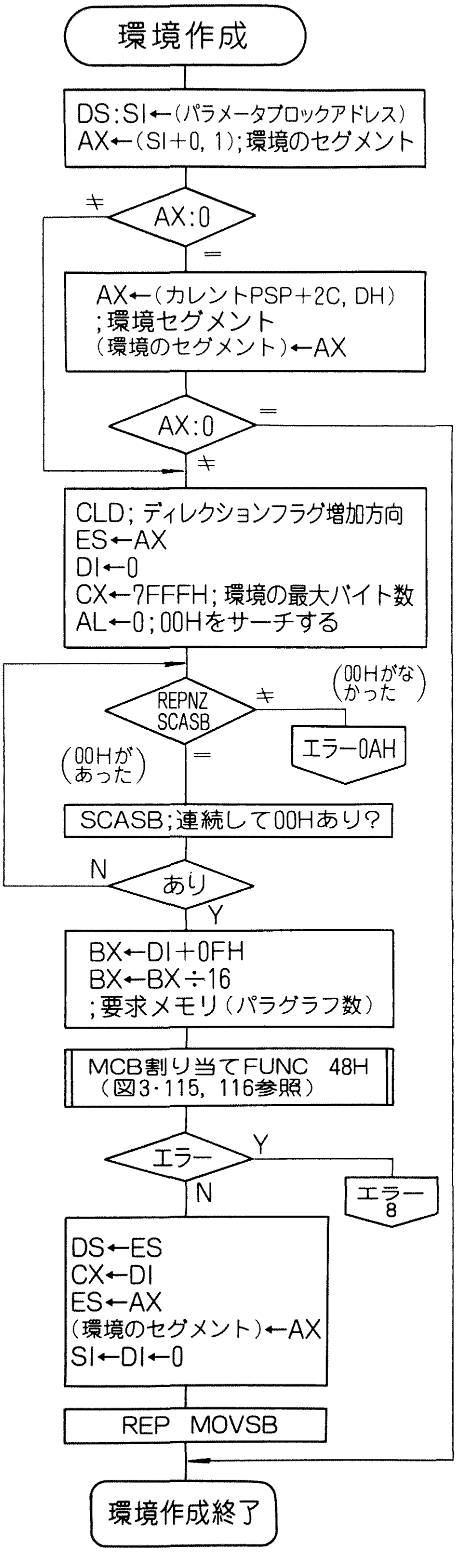


図 3・125 ファンクション 4BH プログラムのロード&実行 解析フローチャート その 4



ファンクション番号	4BH
名称	プログラムロード&実行
入力条件	DS:DX←プログラムパス名 ES:BX←パラメータブロックアドレス AL←0 ロード&実行 (機能コード) 1 ロードしてパラメータブロックにSS:SP, CS:IPの初期値等を返す(実行はしない) 3 ロードのみ
出力条件	CF←0 正常終了 (実行した場合はCFは関係ない) CF←1 異常終了 (エラーコード) AX…1 ALにセットされた機能コードが0, 1, 3以外 2 ファイルが存在しない 実行プログラム名がキャラクタ型 3 指名パス名無効 パスが存在しない 8 メモリ不足 0AH 環境エリアが32kB以上 0BH EXEファイルヘッダが不正 ・ヘッダの最初の1EHを読み込めない ・リロケーションテーブルの先頭へファイルポインタを移動できない ・リロケーション情報を読み込めない

図3・126 ファンクション 4BH プログラムのロード&実行 解析フローチャートと入出力条件 その5

ファンクション4CH
プロセス終了

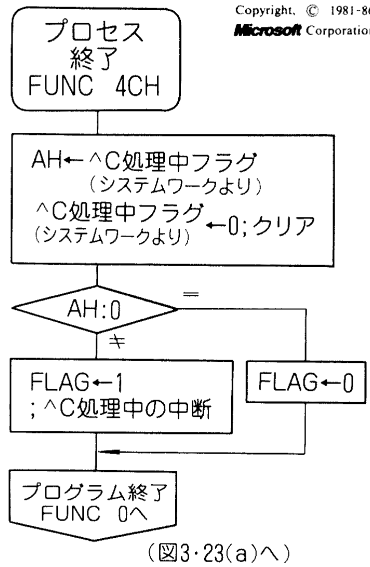


図 3・127　ファンクション 4CH
プロセス終了の解析フローチャート

^C 処理中による終了ならばフラグを 1 にし，それ以外なら 0 にしてプログラム終了共通処理へ飛びます．

ファンクション 00H“プログラムの終了”を見てください．

図 3・127 に解析フローチャートを示します．

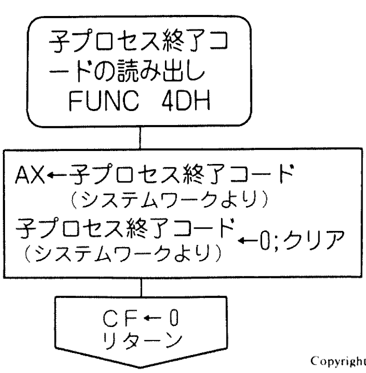
ファンクション4DH
子プロセス終了コードの読み出し

MSDOS．SYS 内の子プロセス終了コードを格納しているシステムワークエリアから読み出すだけです．

終了コードの意味については入出力条件を参照してください．

関連項目：ファンクション 00H“プログラムの終了”
　　　　　ファンクション 4CH“プロセス終了”
　　　　　ファンクション 31H“常駐終了”

図 3・128 に入出力条件と解析フローチャートを示します．



ファンクション番号	4DH
名称	子プロセス終了コードの読み出し
入力条件	な し
出力条件	AL←子プロセスからの終了コード AH…0　INT 20H, FUNC 0, FUNC 4CH による終了 　　1　^Cによる終了 　　2　致命的エラーによる終了 　　3　常駐終了(INT 27H, FUNC 31H)

図 3・128　ファンクション 4DH 子プロセス終了コード読み出しの入出力条件と解析フローチャート

ファンクション4EH
パス名による一致するファイル検索

ファンクション4FH
パス名によるつぎに一致するファイル検索

ファンクション 4E，4FH は検索情報の格納エリアとして DTA を利用します．**図 3・129** にパス名によるファンクション検索時の DTA の内容を示します．

DTA+0～0CH が検索のもとになる情報，+0D～14H がシステム変数，+15H～2AH が検索結果情報となっています．このファンクションを実行するとき DTA の内容をセットする必要はありません．DTA+0～0CH の検索のもととなる情報も検索パス名から，このファンクションが自動的に書き込んでくれます．

ファンクション 4FH は DTA 内のシステム変数エリアをもとにして検索を開始しますので，まえもってファンクション 4EH を実行しておく必要があります．

処理の流れとしては，パス名を解析してディレクトリのクラスタ値とファイル名を求め，ディレクトリをカレントディスクバッファに読み込んで，ファイル名と CL レジスタで指定された属性が一致するディレクトリエントリを見つけます．そのときファンクション 4EH はディレクトリ先頭から検索していくのに対して，ファンクション 4FH は (DTA+0D，EH) の前回までに検索したディレクトリエントリ数のつぎから検索を始めるのです．

DTA	内容		
+0	CL にセットした検索属性		
+1	ドライブ番号（0…カレント）	FCB 形式 検索パス名から作成したもの	
+2～9	ファイル名		
+0A～0CH	拡張子		
+0D，0EH	ディレクトリ先頭から何番目のディレクトリエントリか		システム変数エリア
+0F～12H	検索ドライブ番号に対する DBP アドレス		
+13，14H	検索ディレクトリエントリの存在するディレクトリのクラスタ（0…ルートディレクトリ）		
+15H	検索されたディレクトリエントリの属性		
+16，17H	検索されたディレクトリエントリの最終変更時間		
+18，19H	検索されたディレクトリエントリの最終変更日付		
+1A～1DH	検索されたディレクトリエントリのファイルの大きさ		
+1E～2AH	検索されたディレクトリエントリのファイル名をパックネームにしたもの		

図 3・129　パス名による検索時の DTA

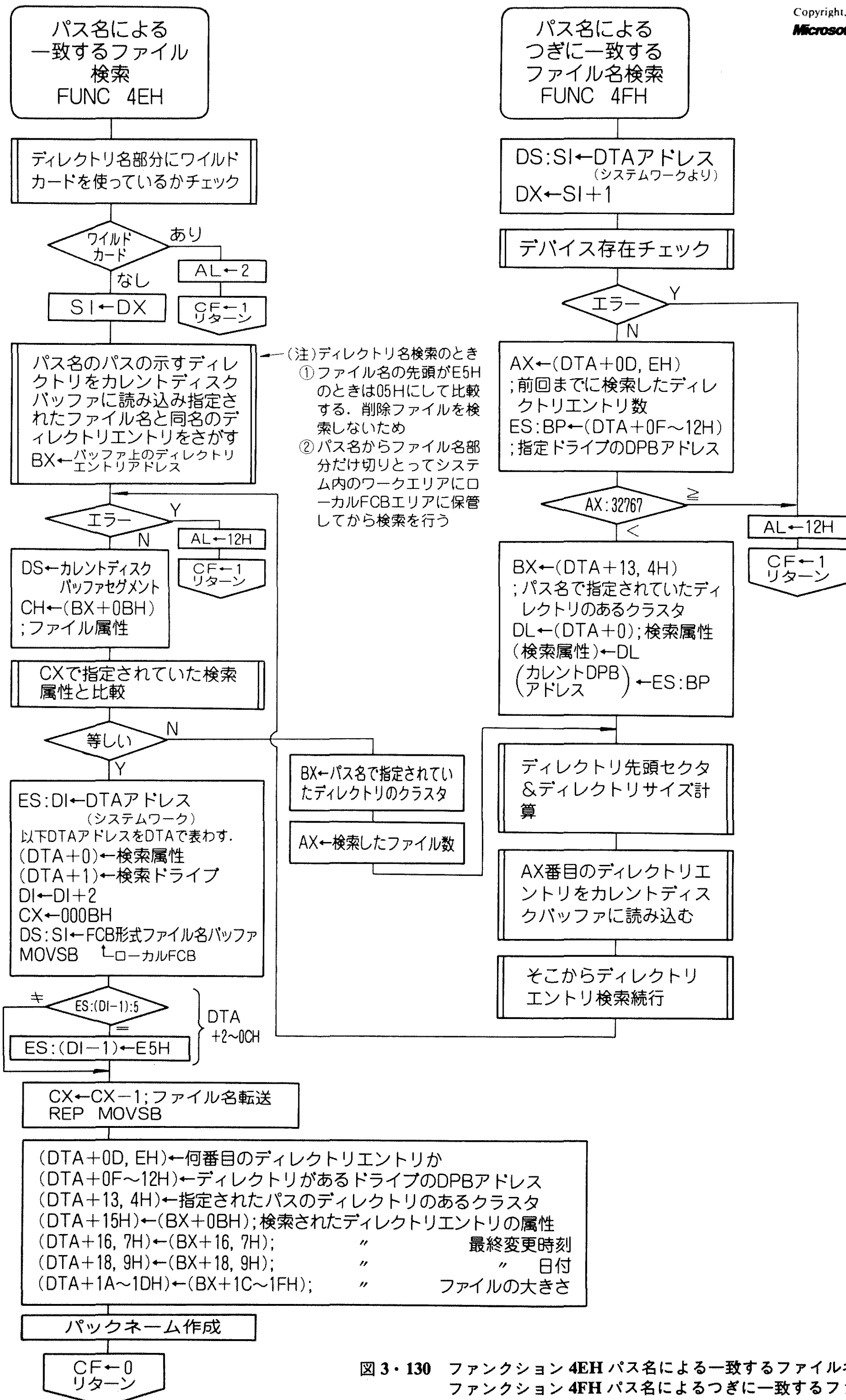


図 3・130 ファンクション 4EH パス名による一致するファイル名検索
ファンクション 4FH パス名によるつぎに一致するファイル検索の解析フローチャート(その 1)

パス名解析においてパス名は

A：¥FOO¥BAR¥FILENAME．EXT

- FILENAME．EXT … DTA＋01～0CH
- BAR … サブディレクトリのクラスタ値（ルートディレクトリのクラスタ値は 0000H となる）
- A … DPB アドレス

と分解されます．

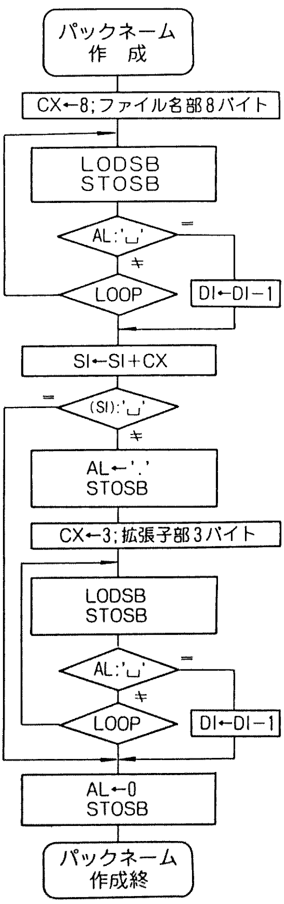
検索が成功すれば，ディレクトリエントリの属性，最終変更時刻，最終変更日付，ファイルの大きさなどを DTA にセットします．

パス名のファイル名部にワイルドカードを使えば，つぎつぎに一致するファイルを検索することができます．

ファンクション 4EH と 4FH の違いをまとめておくと

ファンクション 4EH	ファンクション 4FH
パス名の示すディレクトリ先頭から検索	(DTA＋0D，EH)の示す前回までに検索したつぎの位置から検索 検索ディレクトリは (DTA＋13，4H) で指定されているディレクトリクラスタ
検索属性は CX で指定されたもの	検索属性は（DTA＋0）に格納されているもの
パス名の示すドライブの DPB 使用	(DTA＋0F～12H）の示す DPB 使用

図 3・130，131 に入出力条件と解析フローチャートを示します．



ファンクション番号	4EH	4FH
名称	一致するファイルの検索	つぎに一致するファイルの検索
入力条件	DS:DX←パス名アドレス CX←検索属性 （注）DTA内にシステム変数を確保するので不用意に DTA内の値を変えないこと	
出力条件	CF←0　正常終了 DTAに検索情報 CF←1　異常終了 （エラーコード） AX…2　ディレクトリ名部にワイルドカードを使っている（FUNC　4EH） 12H　これ以上ファイルが存在しない	

図 3・131　**ファンクション 4EH パス名による一致するファイル名検索**
ファンクション 4FH パス名によるつぎに一致するファイル検索の入出力条件と解析フローチャート（その 2）

ファンクション50H/51H
カレントPSPセグメントの設定／取得

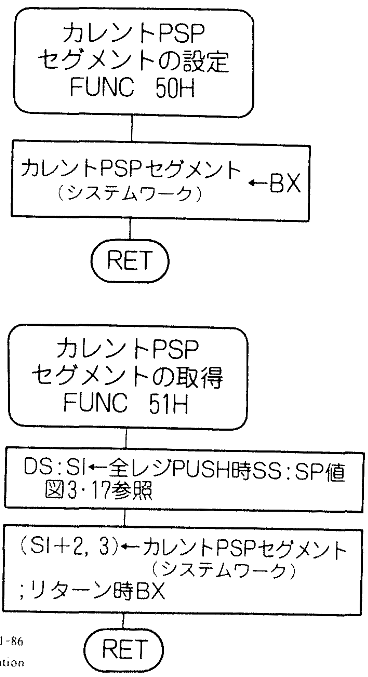
処理としてはMSDOS.SYS内のカレントPSPセグメントを記録しているワークエリアに読み書きするだけです．

カレントPSPについては各ファンクションのカレントPSPの扱いを見てもらえば具体的になんであるかわかりますが，大まかにいえば，COM型ファイルならばPSPの位置はそのプログラムが実行しているCSセグメントの0000H～00FFHに固定されていますが，EXE型になるとそのように固定されていません．したがってMSDOS.SYS内にいま実行されているプログラムのPSPのセグメントアドレス(オフセットアドレスはEXE型，COM型両方とも0000H～00FFHに固定)を記録しておいて，処理がスムーズに運ぶようにしようというものです．

このファンクションを用いればカレントPSPは変更できますが，ファンクション26H，55HのPSP作成を実行したときも自動的にカレントPSPは新規作成したPSPに変更されます．

ファンクションの処理でPSPを使う場合は，ファイルハンドル⟷内部FCB対応表をパス名のオープンやファイルハンドルによるファイルアクセスで使うほか，プログラム終了時（ファンクション0，4CH，31H etc.）終了アドレス，環境へのアクセスなどにも使います．

図3・132に入出力条件と解析フローチャートを示します．



ファンクション番号	50H	51H
名称	カレントPSPセグメントの設定	カレントPSPセグメントの取得
入力条件	BX←設定したいPSPのセグメント	なし
出力条件	なし	BX←カレントPSPのセグメント

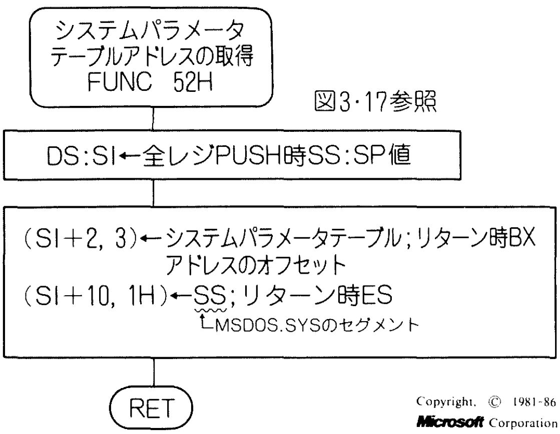
図3・132　ファンクション50H,51H カレントPSPのセグメントの設定/取得の入出力条件と解析フローチャート

ファンクション52H
システムパラメータテーブルアドレスの取得

処理としては，MSDOS.SYS 内のシステムパラメータテーブルのアドレスをリターンするだけです.

システムパラメータテーブルにどういう内容が記録されているかは，**図 3・3, 4** を見てください.

図 3・133 に入出力条件と解析フローチャートを示します.



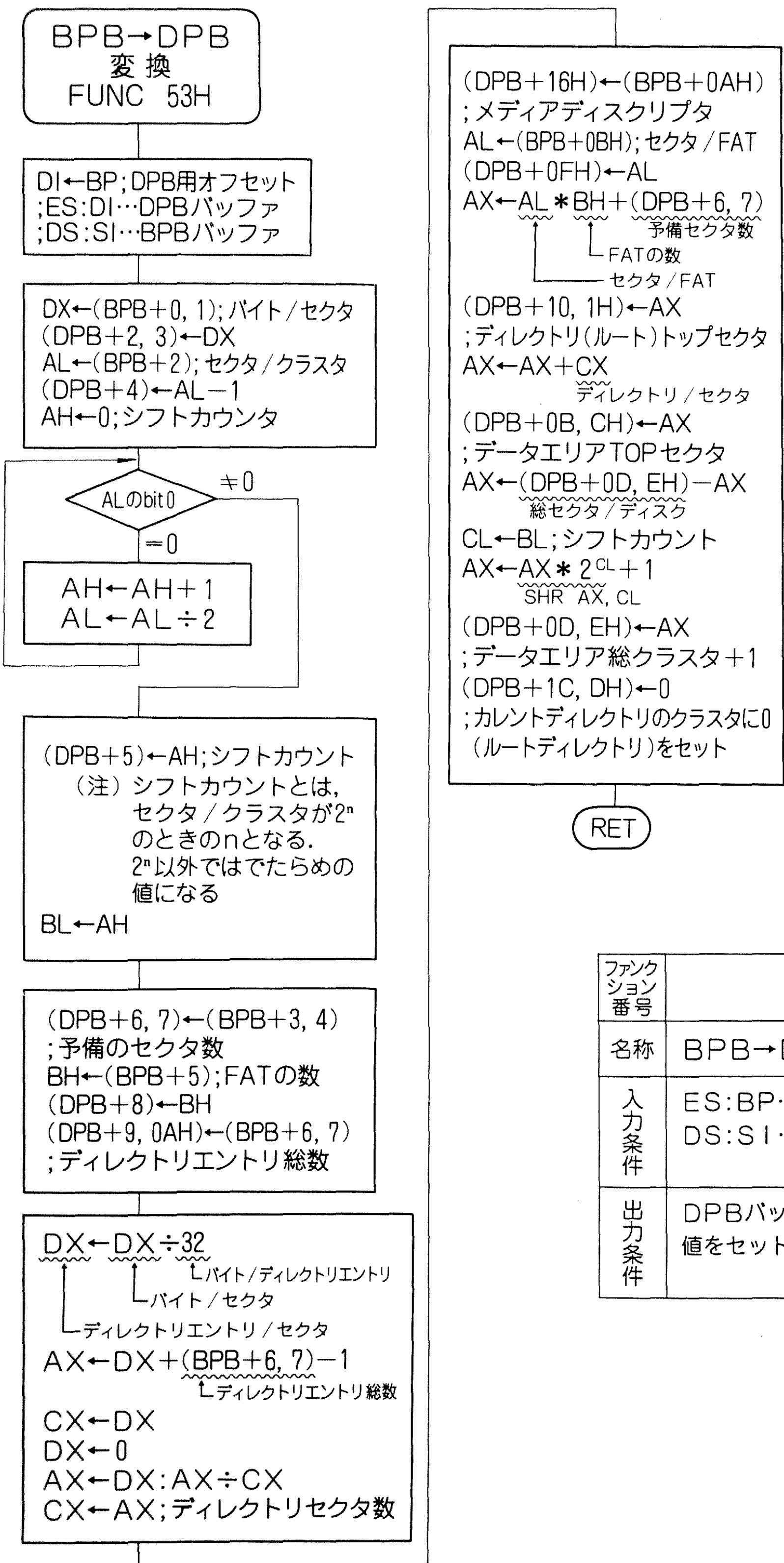
ファンクション番号	52H
名称	システムパラメータテーブルアドレスの取得
入力条件	なし
出力条件	ES:BX←システムパラメータテーブルアドレス（図3・3, 4参照）

図 3・133　ファンクション 52H システムパラメータテーブルアドレスの取得の入出力条件と解析フローチャート

ファンクション53H
BPB→DPB変換

変換方法の説明は，"INT 21H 未公開ファンクションコールの使い方" のほうを参照してください.

図 3・134 に入出力条件と解析フローチャートを示します.



(注)
DPB内
+0…ドライブ番号
+1…論理装置コード
+12～15H…デバイスヘッダへのベクタ
+17H…ディスク交換フラグ
+18～1BH…つぎのDPBへのポインタ
は設定しない.

ファンクション番号	53H
名称	BPB→DPB変換
入力条件	ES:BP…DPBバッファアドレス DS:SI…BPBアドレス
出力条件	DPBバッファにBPBから計算した値をセット

図3・134 ファンクション53H BPB→DPB変換の入出力条件と解析フローチャート

ファンクション54H
ベリファイフラグリード

ファンクション 2EH "ベリファイフラグライト" のほうを見てください.

ファンクション55H
カレントPSPからPSP作成

ファンクション 26H "PSP 作成 (INT コールされたセグメントから)" を見てください.

ファンクション56H
ディレクトリエントリの移動

処理としては, 移動先パス名の指定するディレクトリエントリをあらたに作成し, 移動元パス名のディレクトリエントリの

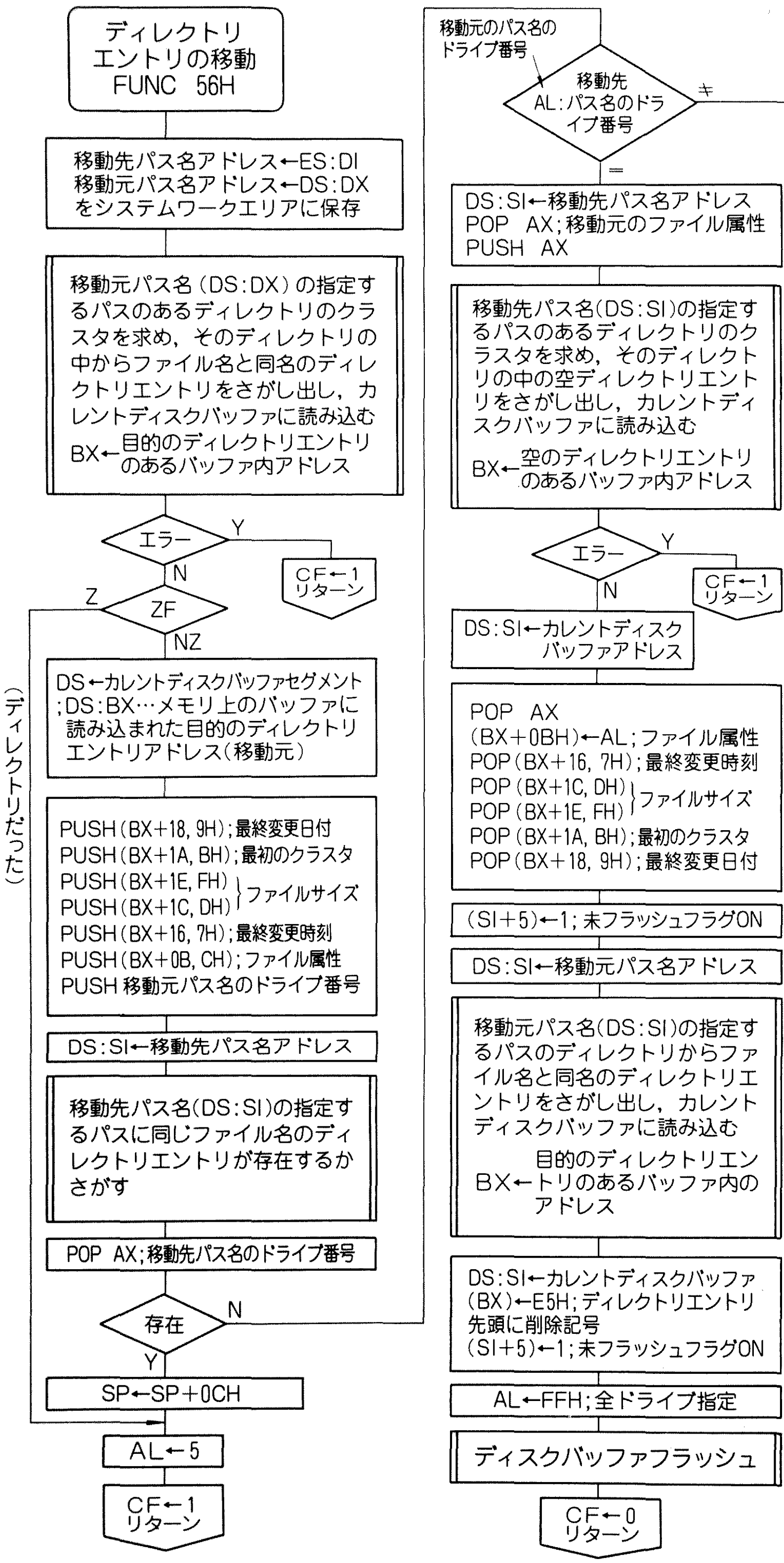
- 最終変更時刻・日付
- 最初のクラスタ
- ファイルサイズ
- ファイル属性

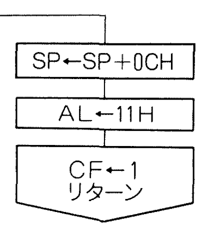
を, あらたに作成した移動先ディレクトリエントリにコピーするだけです.

ただし, 以下の制限があります.

- 移動元パス名が存在すること
- 移動先パス名が存在しないこと
- 移動元パス名がディレクトリでないこと
- 移動元パス名と移動先パス名のドライブが同じであること (ドライブ間の移動はできない)

図 3・135 に入出力条件と解析フローチャートを示します.





ファンクション番号	56H
名称	ディレクトリエントリの移動
入力条件	DS:DX←移動元パス名アドレス ES:DI←移動先パス名アドレス
出力条件	CF←0　正常終了 CF←1　異常終了 (エラーコード) AX…2　移動元パス名(DS:DX)が存在しない 3　指定パス名無効 パスが存在しない 5　移動元パス名(DS:DX)がディレクトリ 移動先パス名(ES:DI)にすでに同名のディレクトリエントリが存在する 11H　移動元パス名のドライブと移動先パス名のドライブが違う (ドライブ間の移動はできない)

図 3・135　ファンクション 56H ディレクトリエントリの移動の入出力条件と解析フローチャート

ファンクション57H
ファイルハンドルによるファイルの時間情報のリード／ライト

ALレジスタに0を指定するとリード，1を指定するとライトになります．ファイル時刻・日付はそれぞれ2バイトで表わされます．

ファイル日付は

bit0～bit4	日付（1～31）
bit5～bit8	月（1～12）
bit9～bit15	年（0～99，1980～2079）

を表わします．年は0～99の値に対して1980を足したものが西暦になります．

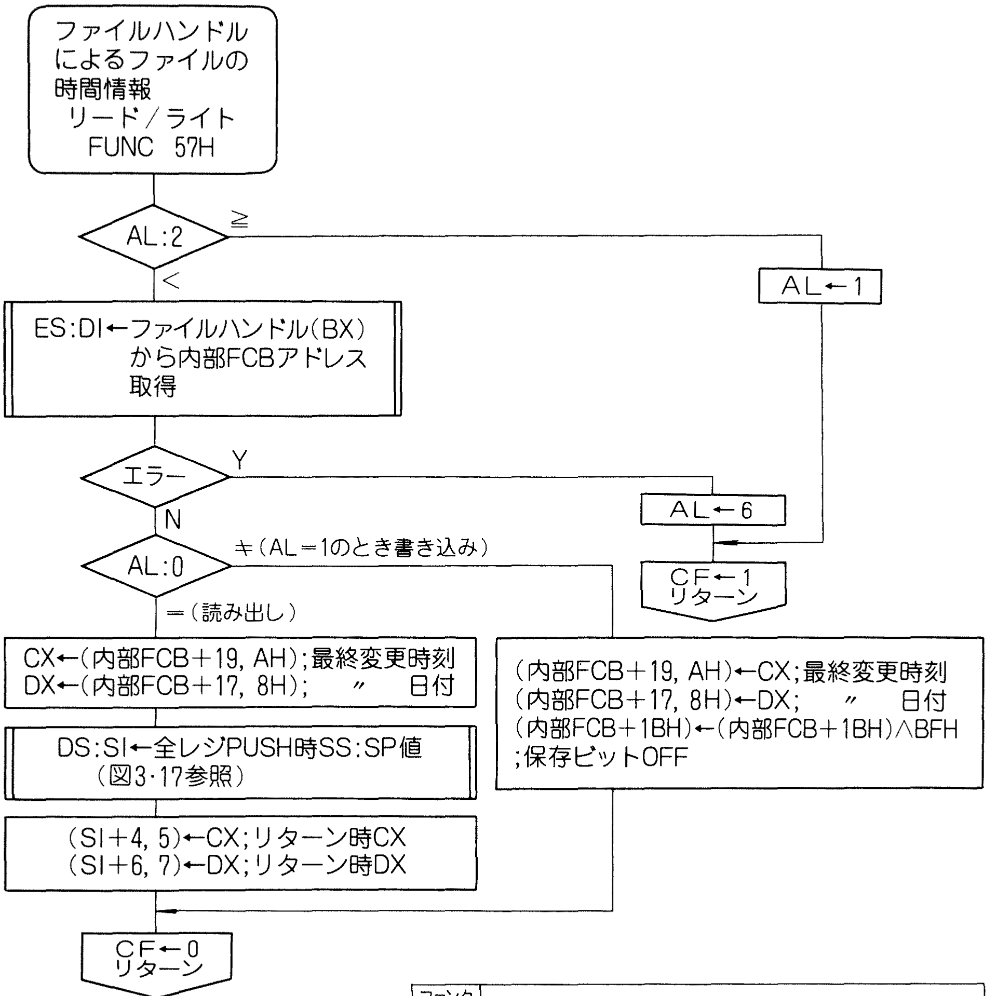
ファイル時刻は

bit0～bit4	秒（0～29，2倍したものが秒）
bit5～bit10	分（0～59）
bit11～bit15	時（0～23）

を表わします．

それぞれのファイルを変更した時点で更新されますので“最終変更時間（タイムスタンプ）”を表わします．

図3・136に入出力条件と解析フローチャートを示します．



ファンクション番号	57H	
名称	ファイルハンドルによるファイルの時間情報 リード／ライト	
入力条件	BX←ファイルハンドル	
	AL←0 リード	AL－1 ライト CX←ファイル時刻 DX←ファイル日付
出力条件	CF←0 正常終了 CX…ファイル時刻 DX…ファイル日付	CF←0 正常終了
	CF←1 異常終了 (エラーコード) AX…1 ALの値が0, 1以外 6 指定ファイルハンドルはオープンされていない	

図3・136 ファンクション 57H ファイルハンドルによる時間情報のリード/ライトの入出力条件と解析フローチャート

ファンクション58H
MCB割り当て法フラグ リード/ライト

MCB割り当て時このフラグによって割り当てアルゴリズムが若干変更されます．このフラグが

0 … できるだけ最下位アドレスに割り当てる

1 … 最も小さいメモリブロックに割り当てる

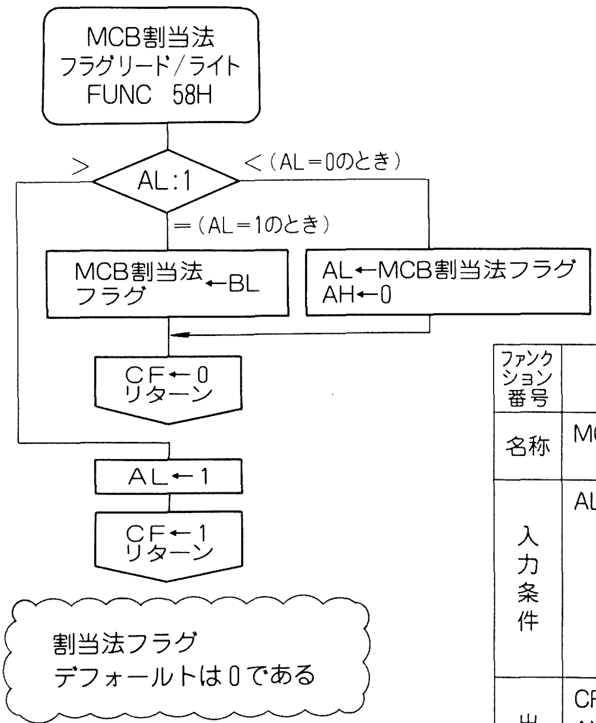
2 … できるだけ最上位アドレスに割り当てる

となっています．具体的な割り当て法はファンクション48H“MCB割り当て”(**図3・115, 116**)を見てください．

このファンクションとしてはそのフラグをリード・ライトするだけです．

ALレジスタが0のときリード，1のときライトとなっています．

図3・137に入出力条件と解析フローチャートを示します．



ファンクション番号	58H	
名称	MCB割当法フラグ リード/ライト	
入力条件	AL←0 リード	AL←1 ライト BX←割当法フラグ 0…最小アドレス 1…最小サイズのフリーMCB 2…最上位アドレス
出力条件	CF←0 正常終了 AX←割当法フラグ	CF←0 正常終了
	CF←1 異常終了 (エラーコード) AX←1…ALに0, 1以外を指定した	

図3・137 ファンクション**58H MCB**割り当て法フラグリード/ライトの入出力条件と解析フローチャート

INT 0

除算エラー処理

DIV または IDIV 命令で 0 で割った場合，INT0 が自動的に実行されプログラムは終了します(もちろん INT 0 を直接実行しても同じ)．

処理としては，CON デバイスへ I/O リクエストコールを直接実行して“0 で除算をしました”と出力し，ファンクション 4CH を AL＝00H にして実行しプログラムを終了します．

INT 20H

プログラム終了

AH＝00H として INT 21H のエントリへジャンプするだけです．**図 3・16** を参照してください．

INT 25H, 26H
物理セクタ　リード/ライト

処理としてはI/Oリクエストコールのコマンド4と8を実行するだけです．このINTコールの処理中もINT 21H処理中フラグ（IN-DOSフラグ）はONにされます．

INTコールに対してRETFで終了しているので，エントリ時のフラグがスタック上に残っています．このINT 25H，26HをしたあとはPOP AXをするなどして，スタックを調整してやらなければなりません．

入出力条件を示しておくと

（入力条件）

AL	…	ドライブ番号（0：カレントドライブ）
DS：BX	…	データバッファ先頭アドレス
CX	…	リード/ライトするセクタ数
DX	…	開始セクタ番号

（出力条件）

CF	…	0　正常終了
		1　異常終了（ALにエラーコード，**図2・9**参照）
CX	…	残りセクタ数

INT 27H
プログラムの常駐終了

COM型プログラムを対象としています．DXレジスタに常駐するバイト数を入れて実行するようになっています．

内部処理としては

DX ← DX＋0FH

CL ← 4

SHR DX，CL

として，INT 21Hのファンクション31Hを（AX＝3100Hとして）実行するだけです．

FATの位置とディレクトリの位置の求め方

これらの情報はDPBにあります．まず，ファンクション1FHかファンクション32Hで，目的のドライブのDPBアドレスを求めます（DPBの先頭アドレスを以下DPBと表わします）．FATの先頭位置（セクタ）は

(DPB+6, 7)　「セクタ]

で求まります．1つのFATの大きさは(DPB+0FH)セクタです．FATは，同じものが(DPB+8）だけ連続して用意されています．したがって，FATは

(DPB+6, 7)～(DPB+0FH)＊(DPB+8)　「セクタ]

を占めていることになります．

つぎに，ディレクトリですが，FATのすぐうしろに位置していますので，ディレクトリの先頭セクタは

(DPB+6, 7)＋(DPB+0FH)＊(DPB+8)　「セクタ]

となります．ディレクトリの最終セクタは，これにディレクトリの大きさを足したものになりますが，もっと簡単に，ディレクトリのすぐうしろがデータエリアの先頭であることを利用して

(DPB+0B, CH)－1　[セクタ]

としてもOKです．ディレクトリの大きさは，(DPB+9, AH)のルートディレクトリのディレクトリエントリの総数を使って

(DPB+9, AH)＊32　[バイト]

または

(DPB+9, AH)＊32/(DPB+2, 3)　[セクタ]

として求まります．

第4章

応用プログラム

4・1　N88日本語BASIC(86)でMS-DOSを20行にする

N88日本語BASIC（86）（MS-DOS版）はMS-DOSとは独自に画面モードの設定を行うので

```
WIDTH  80, 20
```

を実行したあとでMS-DOSのコンソール入出力ファンクションコールを使用したマシン語をCALL（またはUSR(0)）すると，正常に画面表示されません．これはWIDTH　80，20がN88日本語BASIC(86)の画面管理ルーチンのみを20行に設定し，MS-DOSの画面管理ルーチンを20行に設定しないためです．

したがって正常に画面表示させたいならば，MS-DOSの画面管理ルーチンのほうも20行モードにしてやればよいのです．そのためには，つぎのようにDEBUGで短いマシン語プログラムを作り，BASICからBLOAD，CALLしてやればOKです．

```
A>DEBUG
-A
0100  PUSH  DS
      MOV   AX, CS
      MOV   DS, AX
      MOV   DX, 80
      MOV   AH, 9
      INT   21H
      POP   DS
      IRET
-E180   1B  '>3h$'  ;  20行モードセットエスケープシーケンス
-RCX
CX  0000 : 100
-RDX
DX  0000 : 0
-NW20. BIN
-W
-Q
A>
```

BASICからの実行方法は

```
CLEAR  16,   'W20. BINパラグラフ数
DEF  SEG=SEGPTR(2)
BLOAD  "W20. BIN"
```

```
A=0:CALL A
```

です．ちなみに25行にするときは，DEBUG入力を

```
-E180 1B '>3h$'
```

のところを

```
-E180 1B '>3l$'
```

そしてファイル名設定

```
-NW20.BIN
```

を

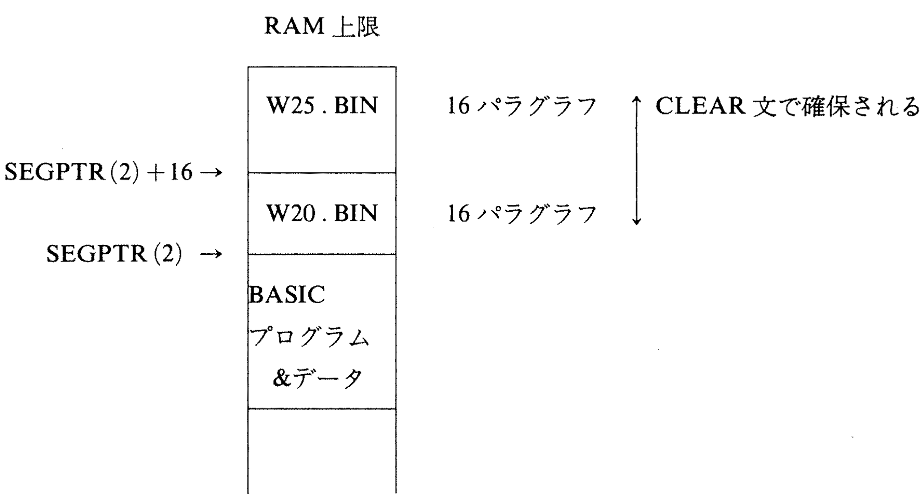
```
-NW25.BIN
```

とでもすればよいでしょう（エスケープシーケンスについては図**2・16**を参照のこと）．

また W20.BIN と W25.BIN を同時に使う場合は，BASIC プログラムを

```
CLEAR 32 'W20.BIN と W25.BIN のパラグラフ数の和
DEF SEG=SEGPTR(2)
BLOAD "W20.BIN"
DEF SEG=SEGPTR(2)+16
BLOAD "W25.BIN"
        :
'W20.BIN を呼び出すとき
DEF SEG=SEGPTR(2)
A=0:CALL A
        :
'W25.BIN を呼び出すとき
DEF SEG=SEGPTR(2)+16
A=0:CALL A
```

とすれば OK です．マシン語プログラムを複数使うときの参考にしてください．



4・2　外部コマンド引数のファイル名のドライブ番号の正しさをすばやく判断する

たとえば

A＞DEBUG　B：FILE1　C：FILE2

外部コマンド名　ファイル名1　ファイル名2

で外部コマンドに入ると，AXレジスタには

AL ……　ファイル名1のドライブ名が正しいとき0，正しくないときFFH

AH ……　ファイル名2のドライブ名が正しいとき0，正しくないときFFH

が入れられて，外部コマンドが実行されます．したがって外部コマンドの先頭で

```
OR    AH, AL
JNZ   ドライブ名エラー処理
```

とするだけで，引数ファイル名のドライブ名部の正しさを判断することができます．

ためしに

```
A>DEBUG  DEBUG  A:X  Z:Y
File  not  found
-R
```

（注）―― 部キー入力

としてごらんなさい．

AX＝FF00 ……

と表示されることでしょう．ドライブ名A：とZ：を逆にして同じことをしてみてください．

AX＝00FF ……

と表示されることでしょう．ドライブZ：が正しくないドライブ名だからです（ドライブ名が接続されているドライブ数を越えているかA～Pの範囲を越えていると，正しくないと判断される）．

この種明かしは，COMMAND.COMが外部コマンドを実行するとき，INT 21Hファンクション4BHを使用しますが，このファンクションの中で上記ファイル引数のドライブ名をチェックしてAXレジスタに正しいとき0，正しくないときFFHをセットするためです．くわしくは，INT 21H解析の節のファンクション4BHを読んでください．

4・3　接続ドライブ数を知る法

マシンに接続されているディスクドライブ数（RAMディスク等も含む）を，簡単に知る方法があ

ります．INT 21H のファンクション 0EH（カレントドライブの選択）を使うのです．このファンクションはカレントドライブの変更と同時に，AL レジスタに接続されているドライブ数を返すのです．カレントドライブを変更せずに接続ドライブ数を得るには，DL レジスタに FFH（ドライブは A～P の 16 台までしか接続できないことになっているので，アスキーコードで 'Q' より大きいものならよい）をセットして実行すれば，AL レジスタに接続ドライブ数だけをリターンしてくれます．

具体的には

```
MOV   AH, 0EH
MOV   DL, 0FFH
INT   21H
```

とすれば OK です．

4・4 アペンド・オープン

ディレクトリにファイルを作るときは ”ファイルの作成（クリエイト）(ファンクション 16H … FCB，ファンクション 3CH …パス名）”，すでにディレクトリ上にあるファイルをオープンするときは ”ファイルのオープン（ファンクション 0FH … FCB, ファンクション 3DH …パス名）”を使います．しかしオープンしたときにはファイルポインタがファイルの先頭にセットされるため，アペンド・オープンにはなりません．

アペンド・オープンとは，以前に作られたファイルのうしろにさらにデータをつぎたしてゆくことです．そのためには，オープンしたあと，ファイルポインタをそのファイルの最後に持ってくれば OK です．ファイルポインタを移動するファンクションとしては，ファンクション 42H ”ファイルポインタの移動（ファイルハンドル用）”があります．

まず，ここではパス名とファイルポインタによる方法を紹介します．COM 型プログラムのときは

```
MOV     DX, offset  FILEBUF
MOV     AL, 2       ;  リード/ライトオープン
MOV     AH, 3DH     ;  ファイルオープン
INT     21H
MOV     BX, AX
MOV     CX, 0
MOV     DX, CX
MOV     AL, 2       ;  ファイルエンド指定
MOV     AH, 42H     ;  ファイルポインタ移動
INT     21H
```

というように，オープンとファイルポインタ移動を組み合わせればアペンドオープンが実現できます．ただし，ここでは

```
FILEBUF  DB  'FILENAME.DAT',0
```

というように，ファイル名がセットされているとしています．EXE 型では DS：DX にファイル名バッファアドレスをセットすれば OK です．

FCBで行うときは'ファイルサイズの計算（ファンクション23H）'を実行したあと'ランダムライト（ファンクション22Hまたは28H）'で書き込みを行えばアペンドになります．

4・5 デバイスの種類

MS-DOSではユーザーがデバイスドライバを定義できますが，デバイスとはなにか，ここで頭を整理しておきましょう．つぎの図を見てください．

デバイス

	キャラクタ型	ブロック型
標準でついてくるドライバの例	PRN，CON，CLOCK，AUX	ディスク
ユーザー定義型のドライバの例	日本語入力フロントエンドプロセッサ　VJE，ATOK5など	RAMディスク IBM-PCと同じフォーマット IBM-PCと異なるフォーマット

デバイスは，大きく分けて"キャラクタ型"と"ブロック型"に分かれます．ブロック型はおもにディスク関係のドライバです．したがってデバイスというと，すぐキャラクタ型のCONやPRNを思い浮かべる人もいると思いますが，ブロック型としてディスクもデバイスの一種だということも頭に入れておいてください．だからRAMディスクなどのディスクのデバイスドライバが定義できるのです．キャラクタ型はCON, PRNなどの名前で管理し，ブロック型はドライブ名（A：, B：, C：など）で管理しているので一見両者はまったく関係ないもののようですが，どちらもデバイスなのです．

4・6 デバイスドライバで処理機能なしの扱い

デバイスドライバで，たとえばPRNなどは出力機能しかないので，データリードコマンド（コマンド番号4）はステータスのDONEビットを立てるだけでリターンしていますが，これでは誤ってPRNを入力に使ったときにでたらめなデータを出してしまい，ときにはシステムダウンにもつながりかねません．こころみに

```
A>COPY  PRN  CON
```

としてください.

これを防ぐためには, I/O リクエストコールのコマンド 0～12 のうち, 処理機能のないものは単にDONE ビットを立ててリターンするだけではなく, それなりに意味のある処理をしてリターンしなければなりません.

たとえば, 上記 PRN の例でいえば, ”データリード”コマンド処理ルーチンは

```
MOV    DS, コマンドパケットセグメント
MOV    BX, コマンドパケットオフセット
MOV    CX, 12H [BX] ;入力データバイト数
LES    DI, 0EH [BX] ;入力バッファアドレス
MOV    AL, 1AH  ;EOF
REP    STOSB
MOV    AX, 100H  ;DONE ビットを立て, エラーなし
         :
         : (あとリターン処理)
         :
```

のように EOF (End of File) のコード 1AH を出力してやればよいのです.

ちょっとしたことですが, 見過ごされがちな重要事項でしょう.

4・7 クラスタ値→セクタ値変換

クラスタ値からセクタ値を計算する方法は次式によります.

(セクタ値)＝(クラスタ値－2)＊(セクタ/クラスタ)＋(データエリア先頭アドレス)

クラスタ値から 2 を引くのは, クラスタ値が 0 からではなく 2 から始まるからです. この式の中で (セクタ/クラスタ) と (データエリア先頭クラスタ) は, DPB の中に格納されていますので, クラスタ値からセクタ値を計算するサブルーチンはつぎのようになります.

ただし, 入出力条件は

(入力)

DL …… ドライブ番号 (0 ……カレントドライブ etc.)

SI …… クラスタ値

(出力)

SI …… セクタ値

です.

```
CLUSTER_SECTER  PROC
  PUSH  CX
  PUSH  DS
  PUSH  BX
```

```
    PUSH  AX
    MOV   AH, 32H       ;DPB アドレス取得
    INT   21H
    MOV   CL, 5 [BX]    ;セクタ/クラスタ　シフトカウンタ
    SUB   SI, 2         ;クラスタ値ーオフセット
    SHL   SI, CL        ;＊(セクタ/クラスタ)
    ADD   SI, 0BH [BX]  ;＋(データエリア先頭セクタ)
    POP   AX
    POP   BX
    POP   DS
    POP   CX
    RET
CLUSTER_SECTER  ENDP
```

上記サブルーチンで計算したセクタ値は，INT 25H（物理セクタによるディスクリード）やINT 26H（物理セクタによるディスクライト）のセクタ値として使えます.

第5章

解析に役立つ
ユーティリティ

5・1　メモリマップ出力　MEMORY.ASM

INT ベクタや IO.SYS, MSDOS.SYS, COMMAND.COM などのセグメント，RAM 上限やフリーメモリ・パラグラフ数などを表示します．

リストは**リスト 5・1** “MEMORY.ASM” を見てください．

このリストの中で “BINHEX.ASM” が INCLUDE されていますが，これを**リスト 5・2** に示します．この BINHEX.ASM は他のプログラムでも使用します．

実行結果を**リスト 5・3** に示します．

COM ファイルの作り方は

```
A>MASM MEMORY;
A>LINK MEMORY;
A>EXE2BIN MEMORY MEMORY.COM
```

ですが，つぎのようなバッチファイルを作っておくとよいでしょう．

ASM. BAT

```
MASM %1;
LINK %1;
DEL %1.OBJ
EXE2BIN %1  %1.COM
DEL %1.EXE
```

すると，

```
A>ASM MEMORY
```

とするだけで，MEMORY.COM ができます（もちろん，リストを正しく入力していれば）．

以降のプログラムも同様です．

```
;
; Display Memory Map
;

code segment

print Macro S               ; Print strings
mov     dx,offset S
mov     ah,9
int     21h
endm

display Macro X             ; Print HEX$(X)
mov     ax,X
call    PrintWord
endm

assume cs:code,ds:code

        org 100h

ramend  = 2
lf  .   = 10
```

```
cr      = 13
dol     = '$'

memory  proc near

display 0                   ; INT vector address
print   message1
display 40h                 ; PC-9800 system work
print   message2
display 60h                 ; IO.SYS segment
print   message3
mov     ax,3521h            ; Get INT 21h vector address
int     21h                 ; is MSDOS.SYS segment
push    bx
display ES
print   message4
pop     bx
display BX
print   message5
mov     ax,352Eh            ; Get INT 2Eh vector address
int     21h                 ; is COMMAND.COM segment
display ES
print   message6
mov     ax,ds:ramend        ; PSP+2,3 is RAM end
push    ax
call    PrintWord
print   message7
pop     ax                  ; RAM end
mov     bx,cs               ; This program segment
sub     ax,bx               ; difference
call    PrintWord           ; is a free memory
print   message8

mov     ax,4c00h
int     21

memory  endp

include binhex.asm

message1 db " INT vectors",cr,lf,dol
message2 db " PC-9800 system work",cr,lf,dol
message3 db " IO.SYS segment",cr,lf,dol,
message4 db " MSDOS.SYS segment",cr,lf,dol
message5 db " INT 21h entry ( offset )",cr,lf,dol
message6 db " COMMAND.COM segment",cr,lf,dol
message7 db " RAM end segment",cr,lf,dol
message8 db " paragraph is a free memory",cr,lf,dol

code    ends

        end     memory
```

リスト5・1 MEMORY.ASM プログラム

```
.XLIST

PrintByte proc near
; Input AL
push    ax
mov     cl,4
shr     al,cl
call    PrintNibble
pop     ax
PrintNibble:    ; Of AL
and     al,0fh
cmp     al,9
jbe     PrintNibble10
```

```
add     al,7
PrintNibble10:
add     al,'0'
mov     dl,al
mov     ah,2
int     21h
ret
PrintByte endp

PrintWord proc near
; Input AX
push    ax
mov     al,ah
call    PrintByte
pop     ax
call    PrintByte
ret
PrintWord endp

PrintDoubleWord proc near
; Input BX:AX
push    ax
mov     ax,bx
call    PrintWord
mov     dl,':'
mov     ah,2
int     21h
pop     ax
call    PrintWord
ret
PrintDoubleWord endp

.LIST
```

リスト **5・2 BINHEX.ASM** プログラム

```
C>memory
0000 INT vectors
0040 PC-9800 system work
0060 IO.SYS segment
0D69 MSDOS.SYS segment
0B19 INT 21h entry ( offset )
1218 COMMAND.COM segment
A000 RAM end segment
8D16 paragraph is a free memory
00
```

リスト **5・3 MEMORY.COM** の実行結果

5・2 MCBリンク表示　MCB.ASM

MS-DOS ではメモリを MCB (Memory Control Block) という単位で管理しています．各 MCB はリンク情報で結合されていますが，この MCB のリンク先頭からリンク終了までのセグメントアドレスや MCB の大きさ等を表示します．

ASM リストを**リスト 5・4** "MCB . ASM" に，実行結果を**リスト 5・5** に示します．

アセンブルするためには**リスト 5・2** "BINHEX . ASM" も必要です．

```
;
; Diplay MCB link
;

code segment

assume cs:code,ds:code

        org 100h

lf      = 10
cr      = 13
dol     = '$'

MCB     proc near

mov     ah,52h          ; System Parameter table address
int     21h
sub     bx,2
mov     ax,es:[bx]      ; Get MCB top segment
xor     si,si
MCBloop:
push    ax
call    PrintWord
call    space
pop     ax
mov     ds,ax
mov     dl,[si]
cmp     dl,'M'  ; M or Z is a normal MCB attribute
je      MCBok
cmp     dl,'Z'
je      MCBok

push    cs      ; else attribute is an abnormal
pop     ds
mov     dx,offset ErrorMessage1
mov     ah,9
int     21h
MCBend:         ; End of Program
mov     ax,4c00h
int     21h

MCBok:
mov     ah,2            ; Print attribute
int     21h
push    dx
call    space
mov     ax,1[si]        ; Print MCB segment
call    PrintWord
call    space
mov     ax,3[si]        ; Print MCB size
call    PrintWord
call    crlf
pop     dx
cmp     dl,'Z'          ; Z is an end of MCB's link
je      MCBend
mov     ax,ds
add     ax,3[si]
inc     ax
jmp     short MCBloop

MCB     endp

crlf    proc near       ; Print CR LF subroutine
push    ax
push    dx
push    ds
```

```
        push    cs
        pop     ds
        mov     dx,offset message1
        mov     ah,9
        int     21h
        pop     ds
        pop     dx
        pop     ax
        ret

message1 db cr,lf,dol

crlf    endp

space   proc near       ; Print space subroutine
        push    ax
        push    dx
        mov     dl,' '
        mov     ah,2
        int     21h
        pop     dx
        pop     ax
        ret
space   endp

include binhex.asm

ErrorMessage1 db 'Break out the MCB$'

code    ends

        end MCB
```

リスト 5・4　MCB.ASM プログラム

```
C>mcb
0D68 M 0008 04AE
1217 M 1218 00C2
12DA M 1218 000A
12E5 M 12EA 0003
12E9 Z 12EA 8D16
```

- 4列目：メモリブロックの大きさ
- 3列目：MCB の属するセグメント
- 1列目：MCB の置かれているセグメント

リスト 5・5　MCB.COM の実行結果

5・3　デバイスヘッダリンク表示　DH.ASM

デバイスヘッダはリンク情報で結合されていますが，そのリンクを先頭からたどって各デバイスヘッダの内容を表示します．属性やストラテジエントリ，インタラプトエントリ，デバイス名またはドライブ数を表示します．

ASM リストを**リスト 5・6** “DH．ASM” に，実行結果を**リスト 5・7** に示します．

アセンブルするためには**リスト 5・2** “BINHEX．ASM” も必要です．

```
C>type dh.asm
;
; Diplay Device Header's Link
;
code segment

assume cs:code,ds:code

print Macro X
push    ds
push    cs
pop     ds
mov     dx,offset X
mov     ah,9
int     21h
pop     ds
endm

lf      = 10
cr      = 13

        org 100h

DeviceHeader proc near

mov     ah,52h
int     21h
push    es
pop     ds
lea     si,17h[bx]
DeviceLoop:
call    crlf
mov     bx,ds
mov     ax,si
call    PrintDoubleWord
call    crlf

print   message1
lodsw
mov     bx,[si]
mov     di,ax           ; save link address
mov     es,bx
add     si,2
call    PrintDoubleWord ; Print link address
print   message2
lodsw                   ; Attribute
push    ax
call    PrintWord       ; Print attribyte
print   message3
lodsw
call    PrintWord       ; Print strategy endtry
print   message4
lodsw
call    PrintWord       ; Print interrupt endtry
pop     ax
test    ax,8000h        ; Is Character type ?
je      BlockDevice     ; No then Block type.
CharacterDevice:
print   message5
mov     cx,8
CharacterDeviceLoop:
lodsb                   ; Print Device name.
or      al,al           ; ASCIZ end ?
jz      NextHeader
mov     dl,al
mov     ah,2
int     21h
loop    CharacterDeviceLoop
```

```
NextHeader:
cmp     di,-1           ; Is End of link ?
je      ProgramEnd
mov     ax,es
cmp     ax,-1
je      ProgramEnd
push    es
pop     ds
mov     si,di
jmp     DeviceLoop
ProgramEnd:             ; Program End.
mov     ax,4c00h
int     21h

BlockDevice:
print   message6
lodsb                   ; Number of disk drives
call    PrintByte
jmp     short NextHeader

DeviceHeader endp

crlf    proc near       ; Print CR LF subroutine
push    ds
push    cs
pop     ds
print   crlfstring
pop     ds
ret
crlfstring db cr,lf,'$'
crlf    endp

include binhex.asm

message1 db       "Next Device Header address  $"
message2 db cr,lf,"Device attribute .......... $"
message3 db cr,lf,"Strategy entry ............ $"
message4 db cr,lf,"Interrupt entry  .......... $"
message5 db cr,lf,"Device name ............... $"
message6 db cr,lf,"Number of disk drives ..... $"

code    ends

        end DeviceHeader
```

リスト 5・6 DH.ASM プログラム

5・4 内部FCB表示 INFCB.ASM

内部 FCB のブロックヘッダと，各内部 FCB のヘッダおよび内部 FCB ファイル名を表示します．ブロックヘッダはリンクアドレスと，そのブロック中にいくつの内部 FCB が存在するかが記録されています．各内部 FCB のヘッダについては，**図 3・9** を参照してください．

ただし，このプログラムは MS-DOS バージョン 2. xx 用です．

ASM リストを**リスト 5・8** “INFCB . ASM” に，実行結果を**リスト 5・9** に示します．

アセンブルするためには**リスト 5・2** “BINHEX . ASM” も必要です．

```
C>dh

091D:0107
Next Device Header address  0F0C:0000
Device attribute .......... 8004
Strategy entry ............ 07D4
Interrupt entry  .......... 07DF
Device name ............... NUL
0F0C:0000
Next Device Header address  0E3E:0000
Device attribute .......... 8000
Strategy entry ............ 0030
Interrupt entry  .......... 0039
Device name ............... MP
0E3E:0000
Next Device Header address  0D69:0000
Device attribute .......... 2000
Strategy entry ............ 0561
Interrupt entry  .......... 0570
Number of disk drives ..... 01
0D69:0000
Next Device Header address  0060:00B4
Device attribute .......... 0000
Strategy entry ............ 09C7
Interrupt entry  .......... 09EC
Number of disk drives ..... 02
0060:00B4
Next Device Header address  0060:00C6
Device attribute .......... 8013
Strategy entry ............ 01B0
Interrupt entry  .......... 01BB
Device name ............... CON
0060:00C6
Next Device Header address  0060:00D8
Device attribute .......... 8000
Strategy entry ............ 01B0
Interrupt entry  .......... 01C1
Device name ............... AUX
0060:00D8
Next Device Header address  0060:00EA
Device attribute .......... 8000
Strategy entry ............ 01B0
Interrupt entry  .......... 01C7
Device name ............... PRN
0060:00EA
Next Device Header address  0060:00FC
Device attribute .......... 8008
Strategy entry ............ 01B0
Interrupt entry  .......... 01CD
Device name ............... CLOCK
0060:00FC
Next Device Header address  FFFF:FFFF
Device attribute .......... 2000
Strategy entry ............ 01B0
Interrupt entry  .......... 01D3
Number of disk drives ..... 02
```

リスト5・7　DH.COMの実行結果

```
;
; Display inner FCB
;        for MS-DOS version 2.XX

code segment

assume cs:code,ds:code

        org 100h

lf      = 10
cr      = 13

print   Macro S           ; Print strings
push    ds
push    cs
pop     ds
mov     dx,offset S
mov     ah,9
int     21h
pop     ds
Endm

space   Macro             ; Print space
mov     dl,' '
mov     ah,2
int     21h
Endm

INFCB   proc near

mov     ah,52h            ; Get system parameter table address
int     21h
lds     si,es:4[bx]       ; In-fcb's link top address
InFcbLoop10:
mov     bx,ds
mov     ax,si
mov     es,bx             ; save link address
mov     di,ax
call    PrintDoubleWord ; Print In-fcb' block link address

call    crlf
mov     ax,4[si]          ; In-fcb Number of this block
call    PrintWord
print   message1          ; FCB's exist in this block
mov     cx,4[si]
add     si,6
InFcbLoop15:
push    cx
mov     al,[si]           ; How many file-handle use this In-fcb
call    PrintByte
space
mov     al,1[si]          ; 0 read-only , 1 write-only , 2 read-write
call    PrintByte
space
mov     al,2[si]          ; attribute
call    PrintByte
space
mov     al,3[si]          ; Drive number
call    PrintByte
space
mov     cx,11
mov     bx,4              ; File name point
InFcbLoop20:
mov     dl,[si][bx]
inc     bx
mov     ah,2
```

```
int      21h
loop     InFcbLoop20
call     crlf
add      si,28h
pop      cx
loop     InFcbLoop15

lds      si,es:[di]        ; Next In-fcb address link
cmp      si,-1             ; Link end ?
jne      InFcbLoop10

mov      ax,4c00h          ; Program end.
int      21h

INFCB    ENDP

message1 db " FCB's exist in this block",cr,lf,"$"

crlf     proc near         ; Print CR LF subroutine
push     ds
push     cs
pop      ds
mov      dx,offset crlfstring
mov      ah,9
int      21h
pop      ds
ret
crlfstring db cr,lf,"$"
crlf     endp

include binhex.asm

code     ends

         end INFCB
```

リスト 5・8 **INFCB.ASM プログラム**

```
C>infcb
091D:0007
0005 FCB's exist in this block
03 02 00 00 AUX
09 02 00 00 CON
03 01 00 00 PRN
00 00 20 03 INFCB   COM
00 02 00 03 DPB2    COM
0F7E:0000
000F FCB's exist in this block
00 00 20 03 BINHEX  ASM
00 00 00 00
00 00 00 00
00 00 00 00
00 00 00 00
00 00 00 00
00 00 00 00
00 00 00 00
00 00 00 00
00 00 00 00
00 00 00 00
00 00 00 00
00 00 00 00
00 00 00 00
00 00 00 00
```

リスト 5・9 **INFCB.COM の実行結果(先頭の数字が 00 のものは未使用内部 FCB)**

5・5 DPB内容表示 DPB.ASM

DPBのリンクを先頭からたどって，その内容を表示します．

MS-DOSバージョン3.xxでは（DPBにカレントディレクトリの格納エリアがないため）カレントディレクトリ関係の表示はしません．

ASMリストを**リスト5・10**“DPB．ASM”に，実行結果を**リスト5・11**に示します．

アセンブルするためには**リスト5・2**“BINHEX．ASM”も必要です．

```
;
; DISPLAY DPB
;

Print macro      ; Print Strings
push    ax
push    ds
push    cs
pop     ds
mov     ah,9
int     21h
pop     ds
pop     ax
        endm

crlf macro       ; Print CR LF
mov     dl,13
mov     ah,2
int     21h
mov     dl,10
mov     ah,2
int     21h
        endm

code    segment
assume  CS:code,DS:code

 FCB equ 5CH

 ORG 100H

DPB     proc near
mov     ah,52h          ; Get system parameter table address
int     21h

lds     si,es:[bx]      ; DPB link top address
DPBLoop:
push    ds
push    si
call    DisplayDPB      ; Display DPB contents.
crlf
pop     si
pop     ds
lds     si,18h[si]      ; Get next DPB link address
cmp     si,-1           ; Link end ?
jne     DPBLoop         ; No then loop.

mov     ax,4c00h        ; Exit
int     21h
```

```
DPB     endp

DisplayDPB proc near
lodsb                               ; Drive number
mov     dx,offset Message1
call    DisplayByte
lodsb                               ; Logic device code
mov     dx,offset Message2
call    DisplayByte
lodsw                               ; Byte/Sector
mov     dx,offset Message3
call    DisplayWord
lodsb                               ; (Sector/Cluster)-1
mov     dx,offset Message4
call    DisplayByte
lodsb                               ; Shift count of Sector/Cluster
mov     dx,offset Message5          ; = Log2(Sector/Cluster)
call    DisplayByte
lodsw                               ; Number of reserve sector
mov     dx,offset Message6
call    DisplayWord
lodsb                               ; Number of FAT
mov     dx,offset Message7
call    DisplayByte
lodsw                               ; Number of route directory endtry
mov     dx,offset Message8
call    DisplayWord
lodsw                               ; Top sector of data area
mov     dx,offset Message9
call    DisplayWord
lodsw                               ; Number of data area Clustor
mov     dx,offset Message10
call    DisplayWord
lodsb                               ; Sector/FAT
mov     dx,offset Message11
call    DisplayByte
lodsw                               ; Top sector of route Directory
mov     dx,offset Message12
call    DisplayWord
lodsw                               ; Vector address of device header
mov     bx,[si]
add     si,2
mov     dx,offset Message13
call    DisplayDoubleWord
lodsb                               ; Media descriptor
mov     dx,offset Message14
call    DisplayByte
lodsb                               ; Disk exchange flag
mov     dx,offset Message15
call    DisplayByte
lodsw                               ; Pointer to next DPB
mov     bx,[si]
add     si,2
mov     dx,offset Message16
call    DisplayDoubleWord
lodsw                               ; Currect directory's cluster

push    ax
mov     ah,30h                      ; Get MS-DOS version number
int     21h
cmp     al,2                        ; Verson 2 ?
pop     ax
jne     DisplayDPBEnd
mov     dx,offset Message17
call    DisplayWord                 ; Print Current directory's cluster

mov     dx,offset Message18
```

```
        call    DisplayPath                 ; Print current directory path name
        DisplayDPBEnd:
        ret
        DisplayDPB endp

        DisplayByte proc near               ; Print HEX byte
        print
        call    PrintByte
        crlf
        ret
        DisplayByte endp

        DisplayWord proc near               ; Print HEX word
        print
        call    PrintWord
        crlf
        ret
        DisplayWord endp

        DisplayDoubleWord proc near         ; Print HEX double word
        print
        call    PrintDoubleWord
        crlf
        ret
        DisplayDoubleWord endp

        DisplayPath proc near               ; Print path name strings
        print
        mov     cx,64
        DisplayPathLoop:
        lodsb
        or      al,al
        jz      DisplayPathEnd
        mov     dl,al
        mov     ah,2
        int     21h
        loop    DisplayPathLoop
        DisplayPathEnd:
        crlf
        ret
        DisplayPath endp

        include  BINHEX.ASM

        message1  db "Drive Number ......................... $"
        message2  db "Logic Device code .................... $"
        message3  db "Byte/Sector .......................... $"
        message4  db "Sector/Clustor-1 ..................... $"
        message5  db "log2(Sector/Clustor) ................. $"
        message6  db "Number of yobi sector ................ $"
        message7  db "Number of FAT ........................ $"
        message8  db "Number of Directory Entry ............ $"
        message9  db "Top sector of Data area .............. $"
        message10 db "Number of Data area Clustor .......... $"
        message11 db "Sector/FAT ........................... $"
        message12 db "Top sector of Directory .............. $"
        message13 db "Vector to Device Header .............. $"
        message14 db "Media descriptor ..................... $"
        message15 db "Disk exchange flag.................... $"
        message16 db "Pointer to Next DPB .................. $"
        message17 db "Clustor of Current directory ......... $"
        message18 db "Path name of Current directory ....... $"

        code    ends

                end     DPB
```

リスト 5・10 DPB.ASM プログラム

```
C>dpb
Drive Number ...................... 00
Logic Device code ................. 00
Byte/Sector ....................... 0200
Sector/Clustor-1 .................. 01
log2(Sector/Clustor) .............. 01
Number of yobi sector ............. 0001
Number of FAT ..................... 02
Number of Directory Entry ......... 0070
Top sector of Data area ........... 000E
Number of Data area Clustor ....... 02CA
Sector/FAT ........................ 03
Top sector of Directory ........... 0007
Vector to Device Header ........... 0060:00FC
Media descriptor .................. 05
Disk exchange flag................. 00
Pointer to Next DPB ............... 091D:444F
Clustor of Current directory ...... 01D8
Path name of Current directory .... opt

Drive Number ...................... 01
Logic Device code ................. 01
Byte/Sector ....................... 0200
Sector/Clustor-1 .................. 01
log2(Sector/Clustor) .............. 01
Number of yobi sector ............. 0001
Number of FAT ..................... 02
Number of Directory Entry ......... 0070
Top sector of Data area ........... 000C
Number of Data area Clustor ....... 027B
Sector/FAT ........................ 02
Top sector of Directory ........... 0005
Vector to Device Header ........... 0060:00FC
Media descriptor .................. 04
Disk exchange flag................. 00
Pointer to Next DPB ............... 0D69:0C88
Clustor of Current directory ...... 0000
Path name of Current directory ....

Drive Number ...................... 02
Logic Device code ................. 00
Byte/Sector ....................... 0200
Sector/Clustor-1 .................. 01
log2(Sector/Clustor) .............. 01
Number of yobi sector ............. 0001
Number of FAT ..................... 02
Number of Directory Entry ......... 0070
Top sector of Data area ........... 000E
Number of Data area Clustor ....... 02CA
Sector/FAT ........................ 03
Top sector of Directory ........... 0007
Vector to Device Header ........... 0D69:0000
Media descriptor .................. F9
Disk exchange flag................. 00
Pointer to Next DPB ............... 0D69:0CE6
Clustor of Current directory ...... 0000
Path name of Current directory ....

Drive Number ...................... 03
Logic Device code ................. 01
Byte/Sector ....................... 0200
Sector/Clustor-1 .................. 01
log2(Sector/Clustor) .............. 01
Number of yobi sector ............. 0001
Number of FAT ..................... 02
Number of Directory Entry ......... 0070
Top sector of Data area ........... 000E
```

```
Number of Data area Clustor ........ 02CA
Sector/FAT ......................... 03
Top sector of Directory ............ 0007
Vector to Device Header ............ 0D69:0000
Media descriptor ................... F9
Disk exchange flag.................. 00
Pointer to Next DPB ................ 0F06:0000
Clustor of Current directory ....... 0000
Path name of Current directory .....

Drive Number ....................... 04
Logic Device code .................. 00
Byte/Sector ........................ 0200
Sector/Clustor-1 ................... 01
log2(Sector/Clustor) ............... 01
Number of yobi sector .............. 0000
Number of FAT ...................... 01
Number of Directory Entry .......... 0100
Top sector of Data area ............ 0014
Number of Data area Clustor ........ 00F6
Sector/FAT ......................... 04
Top sector of Directory ............ 0004
Vector to Device Header ............ 0E3E:0000
Media descriptor ................... F8
Disk exchange flag.................. 00
Pointer to Next DPB ................ 00FD:FFFF
Clustor of Current directory ....... 0000
Path name of Current directory .....
```

リスト 5・11　DPB.COM の実行結果

5・6　ディスクバッファ表示　DBUF.ASM

ディスクバッファのリンク情報をたどって各ディスクバッファのヘッダだけの表示と内容までの表示をすることができます．

A>DBUF

でヘッダのみ

A>DBUF　X

で内容までの表示になります．

ASM リストを**リスト 5・12** "DBUF.ASM" に，"DBUF" としたときの実行結果を**リスト 5・13** に，"DBUF　X" としたときの実行結果を**リスト 5・14** に示します．

アセンブルするためには**リスト 5・2** "BINHEX.ASM" も必要です．

```
;
;  Display disk buffer
;
; Usage : A>DBUF      Display disk buffer link
;         A>DBUF X    Display disk buffer contents
;

code    segment
assume  cs:code,ds:code

        org 100h

DBUF    proc
```

```
mov     si,80h
mov     di,offset flag
movsb
mov     ah,30h
int     21h
mov     si,13h
cmp     al,3
jne     DBUF10
dec     si          ; si<=12h
DBUF10:
mov     ah,52h
int     21h
lds     si,es:[bx][si]
DBUFLoop:
call    dump
cmp     word ptr [si],-1
je      DBUF_END
lds     si,[si]
jmp     DBUFLoop
DBUF_END:
mov     ax,4c00h
int     21h

flag    db 0
DBUF    endp

dump    proc

push    si
lodsw
mov     bx,ax
lodsw
xchg    ax,bx
call    PrintDoubleWord
call    spcl
call    spcl
call    spcl
mov     cx,8
DumpLoop1:
push    cx
lodsb
call    PrintByte
call    spcl
pop     cx
loop    DumpLoop1
lodsw
mov     bx,ax
lodsw
xchg    ax,bx
call    PrintDoubleWord
call    crlf

cmp     cs:flag,0
je      DumpEnd
cmp     cs:flag,1
je      DumpEnd
mov     cx,16
DumpLoop2:
push    cx
mov     cx,16
push    si
DumpLoop3:
push    cx
lodsb
call    PrintByte
call    spcl
pop     cx
loop    DumpLoop3
```

```
call    spcl
pop     si
mov     cx,16
DumpLoop4:
push    cx
lodsb
cmp     al,20h
jae     Dump10
mov     al,'.'
jmp     Dump20
Dump10:
cmp     al,80h
jb      Dump20
mov     al,'.'
Dump20:
mov     dl,al
call    OutChar
pop     cx
loop    DumpLoop4

call    crlf
pop     cx
loop    DumpLoop2
call    crlf
DumpEnd:
pop     si
ret

dump    endp

spcl    proc
mov     dl,' '
call    OutChar
ret
spcl    endp

OutChar proc
mov     ah,2
int     21h
ret
OutChar endp

crlf    proc
mov     dl,13
call    OutChar
mov     dl,10
call    OutChar
ret
crlf    endp

include binhex.asm

code    ends

end     DBUF
```

リスト **5・12 DBUF.ASM** プログラム

```
C>dbuf
11B4:0000    FF 00 00 01 07 00 01 00 0D69:0C88
1172:0000    FF 00 00 01 01 00 02 03 0D69:0C88
1049:0000    FF 00 00 01 CB 01 01 00 0D69:0C88
0FE6:0000    02 00 00 01 CB 01 01 00 0D69:0C88
1193:0000    02 00 E5 01 09 00 01 00 0D69:0C88
1007:0000    02 00 00 01 0A 00 01 00 0D69:0C88
11D5:0000    02 00 E5 01 0B 00 01 00 0D69:0C88
1028:0000    02 00 00 01 0C 00 01 00 0D69:0C88
108B:0000    02 00 00 01 0D 00 01 00 0D69:0C88
```

```
106A:0000   02 00 00 01 CA 00 01 00 0D69:0C88
11F6:0000   02 00 00 01 78 00 01 00 0D69:0C88
0FC5:0000   02 00 00 01 7A 00 01 00 0D69:0C88
110F:0000   02 00 E5 01 7C 00 01 00 0D69:0C88
10CD:0000   02 00 00 01 7E 00 01 00 0D69:0C88
091D:41E1   02 00 00 01 01 00 02 03 0D69:0C88
10AC:0000   02 00 00 01 82 00 01 00 0D69:0C88
1151:0000   02 00 00 01 84 00 01 00 0D69:0C88
0FA4:0000   02 00 00 01 07 00 01 00 0D69:0C88
1130:0000   02 00 E5 01 08 00 01 00 0D69:0C88
FFFF:FFFF   02 00 00 01 AE 00 01 00 0D69:0C88
```

リスト5・13 **DBUF.COM**の実行結果（ディスクバッファヘッダ情報）

```
C>dbuf x
11B4:0000   FF 00 00 01 07 00 01 00 0D69:0C88
4D 58 2D 31 53 74 61 72 28 31 29 08 00 00 00 00   MX-1Star(1).....
00 00 00 00 00 00 52 6E 48 0E 00 00 00 00 00 00   ......RnH.......
49 4E 49 54 20 20 20 20 42 41 54 20 00 00 00 00   INIT    BAT ....
00 00 00 00 00 00 31 A5 95 0D 02 00 F4 02 00 00   ......1.........
43 4F 4D 4D 41 4E 44 20 43 4F 4D 20 00 00 00 00   COMMAND COM ....
00 00 00 00 00 00 6E 82 33 07 03 00 26 43 00 00   ......n.3...&C..
E5 50 42 20 20 20 20 20 41 53 4D 20 00 00 00 00   .PB     ASM ....
00 00 00 00 00 00 ED 91 37 0D 14 00 7B 0B 00 00   ........7...{...
42 49 4E 48 45 58 20 20 41 53 4D 20 00 00 00 00   BINHEX  ASM ....
00 00 00 00 00 00 C4 91 35 0D 17 00 1B 02 00 00   ........5.......
4D 45 4D 4F 52 59 20 20 41 53 4D 20 00 00 00 00   MEMORY  ASM ....
00 00 00 00 00 00 49 71 48 0E 53 00 50 05 00 00   ......IqH.S.P...
4D 43 42 20 20 20 20 20 41 53 4D 20 00 00 00 00   MCB     ASM ....
00 00 00 00 00 00 E6 71 48 0E 55 00 FC 04 00 00   .......qH.U.....
44 48 20 20 20 20 20 20 41 53 4D 20 00 00 00 00   DH      ASM ....
00 00 00 00 00 00 C5 72 48 0E 57 00 06 07 00 00   .......rH.W.....

1172:0000   FF 00 00 01 01 00 02 03 0D69:0C88
F9 FF FF FF 4F 00 05 60 00 07 80 00 09 A0 00 0B   ....O.. ........
C0 00 0D E0 00 0F 00 01 11 20 01 13 F0 FF 00 00   ........ .......
00 00 F0 FF 00 00 00 00 00 00 00 00 00 00 00 00   ................
21 F0 FF 23 F0 FF D9 60 02 FF 0F 00 00 00 00 2B   !..#...`.......+
C0 02 2D E0 02 2F 00 03 31 20 03 33 40 03 35 F0   ..-../..1 .3@.5.
FF FF FF FF FF FF FF FF FF FF 3D F0 FF 3F 00 04   ..........=..?..
41 20 04 43 40 04 45 60 04 47 F0 FF 49 A0 04 4B   A .C@.E`.G..I..K
C0 04 4D E0 04 4F 00 05 51 F0 FF FF 4F 05 FF 6F   ..M..O..Q...O..o
05 FF 8F 05 FF AF 05 FF 0F 00 00 00 00 00 00 06   ................
FF 2F 06 63 40 06 65 60 06 67 80 06 69 A0 06 6B   ./.c@.e`.g..i..k
C0 06 6D E0 06 6F 00 07 71 20 07 73 40 07 75 60   ..m..o..q .s@.u`
07 77 80 07 79 A0 07 7B C0 07 7D E0 07 7F 00 08   .w..y..{..}.. ..
81 20 08 83 40 08 85 60 08 87 80 08 89 A0 08 8B   . ..@..`........
C0 08 8D E0 08 8F 00 09 91 20 09 93 40 09 95 60   ......... ..@..`
09 97 80 09 99 A0 09 9B C0 09 9D E0 09 9F 00 0A   ................
A1 20 0A A3 40 0A A5 60 0A A7 80 0A A9 A0 0A AB   . ..@..`........

1049:0000   FF 00 00 01 CB 01 01 00 0D69:0C88
65 72 20 2E 2E 2E 2E 2E 2E 2E 2E 2E 2E 2E 2E 20   er ............
24 22 0D 0A 6D 65 73 73 61 67 65 31 34 20 64 62   $"..message14 db
20 22 4D 65 64 69 61 20 64 65 73 63 72 69 70 74    "Media descript
6F 72 20 2E 2E 2E 2E 2E 2E 2E 2E 2E 2E 2E 2E 2E   or .............
2E 2E 2E 2E 2E 2E 20 24 22 0D 0A 6D 65 73 73 61   ...... $"..messa
67 65 31 35 20 64 62 20 22 44 69 73 6B 20 65 78   ge15 db "Disk ex
63 68 61 6E 67 65 20 66 6C 61 67 2E 2E 2E 2E 2E   change flag.....
2E 2E 2E 2E 2E 2E 2E 2E 2E 2E 2E 2E 2E 20 24 22   ............. $"
0D 0A 6D 65 73 73 61 67 65 31 36 20 64 62 20 22   ..message16 db "
50 6F 69 6E 74 65 72 20 74 6F 20 4E 65 78 74 20   Pointer to Next
44 50 42 20 2E 2E 2E 2E 2E 2E 2E 2E 2E 2E 2E 2E   DPB ............
2E 2E 2E 2E 20 24 22 0D 0A 6D 65 73 73 61 67 65   .... $"..message
31 37 20 64 62 20 22 43 72 75 73 74 6F 72 20 6F   17 db "Crustor o
66 20 43 75 72 72 65 6E 74 20 64 69 72 65 63 74   f Current direct
6F 72 79 20 2E 2E 2E 2E 2E 2E 2E 20 24 22 0D 0A   ory ....... $"..
```

リスト5・14 **DBUF.COM**の実行結果（**DBUFX**としたときのバッファの内容も表示）

用語総索引

50音順, ABC順

アルファベット順

100 番シリーズ

100番シリーズ

162	BASICコンパイラ活用マニュアル	吉野敏也著	定価2000円
163	Cランニングブック――RUN/C，Lattice C 演習プログラム――	横井与次郎著	定価2500円
164	How to U2――PC-9801U2と親しむ本――	上柿 力著	定価2400円
165	レーザーディスク・ブック	パイオニア監修	定価1800円
166	Super MZグラフィック&サウンドテクニック	小笠原・白石・井村共著	定価2400円
167	パソコン/テレコミ入門――ビジネスマンのためのパソコン電話通信読本――	矢矧晴一郎著	定価2200円
168	最新トランジスタ・アンプ設計法	黒田 徹著	定価2000円
169	THE VM WORLD――PC-9801VMのすべて――	ラジオ技術社編	定価2800円
170	8ミリ・ビデオ撮影テクニック	ラジオ技術社編	定価2000円
171	パソコンビジネスソフト用語事典	秋元龍司著	定価2000円
172	パソコンCAD操縦法入門	竹内銀平著	定価2300円
173	音場を創る――ヤマハDSP-1によるオムニサウンドの世界――	ラジオ技術社編	定価2000円
174	V30機械語ビギナーズガイド	鈴木哲哉著	定価2600円
175	ホップ・ステップ・1-2-3――日本語ロータス1-2-3の使い方――	上柿 力著	定価2000円
176	CCDカメラ技術	竹村裕夫著	定価3000円
177	MS-DOSユーティリティを作る――DOS-BASICとMASMの活用ノウハウ集――	藤田英時著	定価2000円
178	MS-DOSソフト開発環境マニュアル――サポートツールの選び方使い方――	浅野理森著	定価2600円
179	はじめてのパソコン はじめての1-2-3	上柿 力著	定価1800円
180	Prologランニングブック――RUN/Prolog演習プログラム――(上)	横井与次郎著	定価2600円
182	新版ディジタルオーディオ――CD，DATの基礎理論と最新技術――	土井利忠・伊賀章著	定価3000円
183	PC-9800シリーズビギナーズガイドVX編	近藤龍太郎著	定価2000円
184	ビデオ・テクニカル・レポート	編集部編	定価2800円
185	Prologランニングブック――RUN/Prolog演習プログラム――(下)	横井与次郎	定価2800円

500番シリーズ

506	PC110番（PC-9800シリーズ編）	日本電気パソコンインフォメーションセンター監修	定価1800円
507	PC110番（PC-8800シリーズ編）	日本電気パソコンインフォメーションセンター監修	定価1700円
508	MZ110番（シャープMZシリーズ編）	シャープソフトウェアセンター監修	定価1700円
509	FM110番（FM-7/8/NEW7/77編）	富士通マイコンスカイラブ監修	定価1700円
510	X1 110番（シャープX1シリーズ編）	シャープソフト開発部監修	定価1700円
511	Apple110番（II，IIe，IIc編）	アップルコンピュータジャパン監修	定価1800円
513	プリンタ110番（エプソン編）	エプソン株式会社監修	定価1800円
514	続PC110番（PC-9800シリーズ編）	日本電気パソコンインフォメーションセンター監修	定価1800円
515	続PC110番（PC-8800シリーズ編）	日本電気パソコンインフォメーションセンター監修	定価1700円
516	続FM110番（FM-16β/16π編）	富士通マイコンスカイラブ監修	定価1800円
521	続X1 110番（シャープX1，X1turboシリーズ編）	シャープ第4商品企画部監修	定価1700円
523	新PC110番（PC-9800シリーズ編）	日本電気パソコンインフォメーションセンター監修	定価1800円
524	新PC110番（PC-8800シリーズ編）	日本電気パソコンインフォメーションセンター監修	定価1700円
517	パソコン何でも110番	ラジオ技術社編	定価1800円
518	dBASE110番（dBASE II/III）	鈴木良美著	定価2000円
519	ビジネスソフト110番――PC-98用市販ソフト活用ノーハウ集――	横井・岡田・辻共著	定価2000円
520	MS-DOS110番	吉沢一秀著	定価2300円
522	C言語110番	伊藤英治著	定価2300円
512	Z80マシン語ランド	内田直樹著	定価2000円
525	二代目大番頭――パソコンで財務会計をする人のために――	矢野和代著	定価1200円
526	MS-DOS theアプリケーションソフト	ラジオ技術社編	定価1700円
527	松・桐・楓 ファミリー活用術	マイクロデザイン編著	定価1200円
528	MS-Cプログラミングガイド	松坂邦生著	定価1700円
529	一夜漬COBOLプログラミング	大串 信著	定価1200円

発行 ラジオ技術社 東京都千代田区神田淡路町2-1 TEL 03-251-0498

オーディオ技術関係

著者略歴

幸田敏記（こうだとしき）

1958年，福岡県久留米市に生まれる．

1984年，九州大学大学院総合理工学研究科エネルギー変換工学専攻卒業後，九州電力勤務．はじめて手にしたマイコンは，MP-80というワンボードマイコンだった．パーソナルコンピュータは趣味といえども，コンピュータに関しては幅広い知識を持つ．おもな著書に

PC-TechKnow9800（共著，システムソフト）

マシン語クックブックI（共著，システムソフト）

マシン語マジックブックI（共著，日本ソフトバンク）

マシン語クックブック MS-DOS（BNN）

がある．

ISBN4-8443-0181-0 C2055 ¥3500E

MS-DOS データブック

— MSDOS.SYS の仕組みと解析フローチャート —　定価 3,500円

昭和62年11月20日　初 版 発 行

検 印
省 略

Printed in Japan

著者© 幸 田 敏 記
編集人 鈴 木 勇 治
発行人 金 井 稔

発行所 株式会社 ラジオ技術社
〒101 東京都千代田区神田淡路町2－1
☎03（251）0498，1241〈振替・東京5-2506〉

中央印刷　トキワ製本

ラジオ技術選書181

ISBN4-8443-0181-0 C2055 ¥3500E

定価 3,500円